MACHINE INDEPENDENCE: THE PROBLEM OF PORTABILITY

Edward Carl Coulter

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

MACHINE INDEPENDENCE
THE PROBLEM OF PORTABILITY

by

Edward Carl Coulter
and
Daniel James Parker

June 1974

Thesis Advisor: G. A. Kildall

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO. / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Machine Independence The Problem of Portability	5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; June 1974 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Edward Carl Coulter and Daniel James Parker	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940	12. REPORT DATE June 1974 13. NUMBER OF PAGES 96
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Naval Postgraduate School Monterey, California 93940	15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Portability
Machine Independence

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Computer program portability is a measure of the effort which must be expended when transferring a program from one environment to another. Historically, there have been several approaches to portability, but most have met with little success. The problems of portability are presented, along with techniques available to computer programmers to solve or alleviate those problems. Concurrent with this study, a portable preprocessor/editor was designed. The preprocessor/editor was intended to

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20.

give FORTRAN users a machine-independent means of character output and provide a simple editor for text manipulation.

Machine Independence
The Problem of Portability

by

Edward Carl Coulter
Lieutenant, United States Navy
B.S., University of Arizona, 1967

and

Daniel James Parker
Lieutenant, United States Navy
B.A., San Diego State College, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1974

ABSTRACT

Computer program portability is a measure of the effort which must be expended when transferring a program from one environment to another. Historically, there have been several approaches to portability, but most have met with little success. The problems of portability are presented, along with techniques available to computer programmers to solve or alleviate those problems. Concurrent with this study, a portable preprocessor/editor was designed. The preprocessor/editor was intended to give FORTRAN users a machine-independent means of character output and provide a simple editor for text manipulation.

TABLE OF CONTENTS

LIST OF FIGURES

I. INTRODUCTION

Of major concern to the computer industry in recent years has been what W. M. Waite [Ref. 1] describes as the "Software Crises." That is, the problem of reprogramming when the need arises to transport software from one machine to another must be given major consideration as a factor in computer program portability. The ease which the transfer is accomplished is measured by two factors, portability and adaptability. Portability can be defined as a measure of the ease with which a program can be transferred from one environment to another, while adaptability is a measure of ease with which a program can be altered to fit differing user applications [Poole and Waite, 1973]. Although the differences between the two are not always well defined in practice, the major distinction lies in the above definitions. Where portability deals with environmental changes, such as word length, adaptability is concerned with changes in algorithm structure.

A. RESEARCH GOALS

This research was an initial study of program portability. It was intended to present a broad picture of the problems of software transferability and provide some insight to how those problems could be solved. The programming of a preprocessor/editor was undertaken to provide some insight to the problems of portability while providing a facility to allow machine-independent FORTRAN based software.

B. TERMS AND DEFINITIONS

The following terms and their definitions are provided to give the reader a better understanding of the terminology used.

1. Bootstrapping

As used here, bootstrapping is a process commonly used in compiler implementation. The process is used when a compiler for a source language L (or a subset of L) is implemented on a particular machine and the compiler is to be written in the language L. Gries [Ref. 2] offers the following explanation in the form of an exercise:

> Write a compiler in assembly language for a small subset L[0] of L. This subset should be small, so that it is easy to implement, but large enough to be used in the next step. The next step is to rewrite the compiler for L[0] in itself and check it out. Now, we try to "bootstrap" our way up to L in a series of steps as follows. At each step i, i = 1, ..., n, extend the compiler for L[i-1] to a compiler for a language L[i], by implementing other features of the desired language L.

2. Macro

A macro gives the programmer the ability to refer to a specific code sequence by mentioning a single predefined name. Waite [Ref. 3] defines macros as having a form similar to the following:

```
MACRO NAME (P1,P2,...,Pn)

Code Body

END
```

> The words "MACRO" and "END" serve to delimit the definition, "NAME" is the macro name, and "P1" through "Pn" are formal parameters. The code body is a series of lines which may contain instances of the formal parameters.

Unlike subroutine calls which occur at run time, macro expansion is the result of a preprocessing function. That is, when a macro call is recognized, actual parameters are substituted for formal parameters and the desired code is inserted at the point of the macro call.

3. Translator

McKeeman [Ref. 4] defines a translator as a "function whose domain is a source language and whose range is contained in a target language." Simply stated, it is a mapping of one language into another.

There are many factors which contribute to the problems of program portability. These factors may range from the manufacturer's unwillingness to produce compatible machines to the software community which, as stated by Fleiss, "has not provided the necessary guidelines or standards for program design, coding and documentation" [Ref. 5]. Because of these factors, one finds it necessary to develop portable software. Two approaches to solving the problems of portability are discussed: MACRO processors and high level language programming.

II. MACRO PROCESSORS

Various approaches to portability through the use of macro processors have been attempted in the past. As early as 1961 an attempt to enhance portability was undertaken in a project known as SLANG. This project was somewhat similiar to another early approach, UNCOL. Both systems used translators and were based upon the premise that a translator can be written in a single intermediate language. Poole and Waite [Ref. 6] attribute the failure of these systems to their "simplistic" modeling. The variety of data bases and operators in existing languages necessitates a more comprehensive approach to abstract machine modeling. As Halpern [Ref. 7] points out, "with new computer applications being found daily, with the computer-using population growing at a still accelerating pace, and with that population taking on an increasingly lay character, the notion of a universal language is indefensable today." Since the computer world has somewhat settled on a multiplicity of languages, the above statement seems to hold. Processors on the other hand, are generally tailored to fit one language necessitating a multitude of compilers. The need is for a software processor which can handle the variety of languages which exist today and is somewhat extendable to meet the needs of future languages. To this end, the macro processor has been developed.

A. THE MOBILE PROGRAMMING SYSTEM

The Mobile Programming System is a translator system consisting of two levels, SIMCMP and STAGE2. The system is a "powerful macro processor designed specifically as a tool for constructing machine-independent software" [Ref. 8]. The uniqueness of the Mobile Programming System is that unlike most approaches to portability through macro processors, it does not require an existing version of

itself. The system can be bootstrapped on most contemporary computing machines starting with the simplified translator called SIMCMP.

1. SIMCMP

SIMCMP (simple compiler) is a simple macro processor used to translate any source language which can be defined by macros with single character parameters. The SIMCMP algorithm consists of two distinct phases, macro definition and macro expansion. During the macro definition phase, all macros are read-in using macro names or "templates" as the start of the definition and an end-of-line flag to delimit the definition. Once the definition phase is complete, expansion begins. In the expansion phase, each line of the source code is read. The source statement is matched with the predefined templates. If a match is found, the corresponding macro code body is punched out and the translator continues. If no template match is found for a line of source code, the code is treated as a statement in the base language. Reading a blank line from the source code terminates the process.

2. STAGE2

The second level of the bootstrapping process is STAGE2, consisting of a more powerful processor written in a language capable of being translated by SIMCMP. The purpose of STAGE2 is to provide a means for implementing machine independent software. Software support of STAGE2 has been kept minimal and, as Waite [Ref. 8] pointed out, only five basic functions are required of the supporting machine.

1. read one line
2. detect end-of-file
3. write one line
4. write end-of-file
5. rewind

STAGE2 employs a scanner to recognize macro calls and isolate parameter strings. Template matching is used for recognition of macro calls. Where SIMCMP was a very

simple processor, STAGE2 is a more powerful and flexible processor providing the user many fuctions. The macro facilities of STAGE2 are similar to more powerful macro assemblers, and will be reviewed below for completeness.

a. Storage

STAGE2 provides three levels of storage: parameter storage, associative memory, and up to nine input/output files.

Macros have storage space for nine parameters, with parameter storage local to the macro body. That is, the parameters are available only to the code body of the macro in which they were declared. Because nested macro calls are allowed, parameter values are saved during these calls. At the end of the code body, however, all parameters associated with that macro call are lost and not retrievable.

The associative memory is similar to the COMMON statement in FORTRAN programming. That is, the associative memory area is global to the entire set of macro calls and provides a means for transferring information between macros.

"By suitable processor functions the user can direct output generated by STAGE2 to any file on which writing is allowed. The input may be switched to any file on which reading is allowed" [Ref 8]. Multi-pass operations are possible since the input/output files can be used for temporary storage of intermediate text.

b. Arithmetic Expression Evaluation

Addition, subtraction, multiplication, and division can be performed by STAGE2. All arithmetic evaluations must be of integer type. That is, the operands must be either integers or symbols, where each symbol has an associated integer value stored in associative memory.

c. Conditional and Loop Operations

String equality and the relative magnitude of two expressions can be tested. As a result, an option is

available where the ability to skip a predetermined amount of code is available. The code body of the calling macro may be exited on this type of conditional branch. Looping within a macro call is also available. Similar in structure to a FORTRAN DO-Loop, the calling macros code body may be duplicated more than once.

As pointed out ealier, the Mobile Programming System can be bootstrapped on a target machine without an existing version of itself. SIMCMP is defined in FORTRAN because of the high availability of FORTRAN compilers. However, if a FORTRAN compiler is not available, SIMCMP is easily coded in assembly language on most computers. STAGE2 is written in FLUE (First Language Under Bootstrap) and can be compiled with SIMCMP. After STAGE2 has been implemented in its simplest form, it is "Bootstrapped" until the desired level is reached. As Poole and Waite [Ref. 1] point out, the resulting software is, therefore, tailored "tc the particular problem at hand" and its complexity is extended only "as the need arises."

B. LIMP

LIMP (Language Independent Macro Processor) [Ref. 7] is presently used to process text for eight different compilers. LIMP embodies the concept of a macro processor which is independent of assembly language. Based upon the idea of string manipulation with parameter substitutuicn, it has the capacity to allow the programmer to alter the behavior of a specific compiler without redefining the entire language.

Somewhat similar to the Mobile Programming System in operation, LIMP provides a much broader means of macro definition. Although still referred to as a template, the line which introduces the macro definition may be any arbitrary string of characters. A special parameter symbol is used to indicate the positions of parameters. Waite [Ref. 9] suggests "this approach frees the system from any

arbitrary format restrictions of a particular language - templates can be written in a form suited to the language at hand." Any code line which cannot be matched to a template is copied directly to the output source without change.

1. Parameters

A maximum of nine formal parameters may be specified within the macro code body. They are specified by their relative position in the macro definition template using the integers one through nine. Unlike the usual substitution of actual for formal parameters, LIMP allows several conversion types by indicating the desired subscript on the relative address of the parameter. The conversion digits describe the ways in which the actual parameter string may be used in each substitution. When more than nine formal parameters are needed, a private tree may be used to store intermediate results, as described below.

2. Trees

Three types of trees are available in LIMP: private tree, symbol tree and template tree. As mentioned earlier, private trees may be used to store intermediate results when an excess of nine parameter strings are needed. Except for the macro code body, which is converted to list structure when defined, all input source code is considered in string form. It is, therefore, not possible to use string operations to make changes within the code body. If the user forsees the need to alter the code body of a macro, the code body can be stored in the private tree as a string. It is then possible to redefine this string as the code body of any template.

The symbol tree is used to store symbols which are either created by parameter calls or direct declaration by the user. In conjunction with this, any symbols added to the tree are automatically given the current value of a built-in location counter or explicitly assigned by the user.

The template tree contains the templates of all

defined macros. When an input line is matched to a template, actual parameter strings are created and the associated macro code body is duplicated. The user may reenter the definition phase by using a special command to define another template and add it to the template tree. A blank line after the macro definition causes the system to return to the expansion phase.

The LIMP system does, to some extent, solve the problems of languages being tailored to fit target compilers. It allows flexibility in software which is critical to the problems of transferability. W. M. Waite suggests that, with planned extensions, LIMP could far surpass the role of present macro processors and "would have the power of a syntax directed compiler" [Ref. 9].

Both the Mobile Programming System and LIMP have enjoyed some degree of success. The Mobile Programming System has been implemented on eight different machines with a maximum of two man weeks required for implementation. As Poole and Waite point out, "it is not even necessary to have access to a running version of the system on another machine; a listing, tape or BCD deck is sufficient" [Ref. 1]. LIMP, on the other hand, was not designed with the same goals in mind. The degree of independence of LIMP is directly related to the availability of a WISP compiler on the target machine. WISP is currently available on the IBM 7094, 7040, GE 265, ENGLISH ELECTRIC KDF9, ELLIOT 803, EDSAC 2 and ATLAS 2 computer systems. Once implemented, LIMP is capable of processing text for eight different compilers. As a result, text acceptable to LIMP attains some degree of machine independence.

While the use of macro processors is a valid approach to portability, the development of standardized high-level language compilers opens the door to programming in a high-level language to attain a degree of portability. Software written in high-level languages have been reasonably successful in the field of transferrability.

III. HIGH-LEVEL LANGUAGE PROGRAMMING

The use of high-level languages is evident in many attempts to enhance portability. The macro processors mentioned previously can themselves be highly portable systems if written in a widely available high level language. If this approach is not taken, the job of transferring the system deteriorates to one of reprogramming in assembly language. The alleviation is to code the applications programs by coding directly in universally available high-level languages.

A. STANDARDIZATION

The need for standardization of programming languages is evident. "This opinion is reflected by the existence of standards committees such as ANSI (American National Standards Institute) which have been organized for most popular languages" [Ref. 5]. Industry is very much concerned with maintaining a degree of standardization. "Presently, there are COBOL precompilers that enforce management selected language usage constraints. Information Management Inc., for instance, has MAGIC Standards Enforcer which notes use of management-prescribed COBOL verbs. Arkay Computer Applications' MECCA flags about 200 management selected violations" [Ref. 5]. These standard enforcement packages are largely oriented toward maintaining uniformity of the listings for the sake of readability, but they can flag the use of any selected reserved words. In order to standardize programming languages, in 1969 the UNITED STATES NAVY issued OPNAVINST 10462.8. This instruction explains the Navy's need for standardization and sets forth well documentated standards for the COBOL, FORTRAN, and JOVIAL programming languages. Standardization allows the coding of applications programs within the constraints of a programming language. The implementation of that program on

any target machine is dependent on compiler availability and a minimum consideration of machine-dependent aspects, such as word size, I/O interfaces, character sets, and peripherals. The availability of standardized high-level language compilers is a problem which is rapidly decreasing. As early as 1966, the Navy attempted standardization by issuing SECNAVINST 10462.8 which demands that all new computer acquisitions must specify COBOL and FORTRAN compilers as a mandatory requirement. The Navy realizes that certain high-level languages are better for different user requirements, and, while it limits and standardizes the allowed high level languages, it provides a wide base for running its software.

Standardization is lost when a language is extended to meet certain specialized needs. It seems that even the most powerful high level language compilers are altered to provide extra features deemed necessary by implementors of the compiler. These features are added at the expense of the transferability of programs coded for processing by the "improved" compiler. In time, however, even these extended languages may become standardized. ANSI FORTRAN, for example, is an extension of the original FORTRAN IV language. If extended languages are implemented, their compilers can be written to scan source code for factors which influence portability. "Whenever possible, compilers should flag usage of non-standard language features that they implement. WATFOR ... is an excellent example of fulfilling this function"[Ref. 5].

One successful method of extending a language is to write a preprocessor which converts the extended language into a standardized language. MORTRAN, a language developed at Stanford University, is an example of high-level language extension without significant loss of portability. Written in ANSI standard FORTRAN IV, the MORTRAN processor can be implemented on a wide variety of machines. The MCRTRAN processor translates the MORTRAN input text directly into

FORTRAN statements which can be compiled by the user's FORTRAN compiler. MORTRAN's features include:

1. free field format
2. alphanumeric statement lables
3. comments inserted anywhere in the text
4. multiple assignments
5. implicit looping and block structure
6. statements such as FOR-BY-TO, WHILE, UNTIL, and IF-THEN-ELSE
7. user defined macro-instructions
8. blocks of normal FORTRAN statements inserted in the text

These extensions afford the FORTRAN user a powerful language which retains the qualities of portability inherent in a standardized language such as FORTRAN.

B. STRUCTURED PROGRAMMING

Programming in a standardized, widely available high-level language offers more than just a multiplicity of compilers. The programmer has the opportunity to enhance the transferability of software by using the techniques of structured programming. Structured programming can be thought of as "the goal of evolving a program so that its final structure can be presented or described as a hierarchy of tasks or blocks" [Ref. 10]. Because those areas of computer software which retain machine dependencies must be altered when a transfer of software is to be affected, it is beneficial to the personnel performing the transfer to easily identify those dependencies. Structured, or modular programming, makes it "possible to study the system one module at a time" [Ref. 5] and therefore increases the ease of comprehension of the system. This type of modular programming can be used to segregate those areas in the software which may contain machine dependencies. Knowledge of the entire system or unrelated modules is, therefore, not required to accomplish the task of making the needed changes.

FORTRAN is an excellent example of a high-level language which can be easily modularized. "The ability to divide machine/system dependent and independent attributes into separate entities via sub-programs greatly enhances the transferability of programs" [Ref. 5]. The compiler for INTEL's PL/M language was written in ANSI standard FORTRAN [Ref. 11]. Because the compiler was written using the concepts of structured programming, machine dependencies were easily isolated from the remainder of the system. As a direct result of the modular design and standardized base language, PL/M has been easily implemented on eight different machines with little or no difficulty. Input/output file names are contained within two subroutines and can easily be accessed for change. Once implemented on a specific computing machine, sections of the PL/M compiler may be accessed to increase efficiency. This has been demonstrated on the IBM System/360. A forty percent decrease in compile time was achieved by replacing the FORTRAN method of shifting with an assembly language shift and mask operation. This adjustment, plus the implementation of an assembly language input/output package and the use of the FORTRAN H optimizing complier, allowed compilation of PL/M programs in less than half the time required by the first version. The total effort expended to facilitate these changes was twenty-four man hours.

The advantages of coding in a standardized language similar to FORTRAN are apparent. As Hamlet [Ref. 12] points out, "only schemes based on ANSI FORTRAN really move easily among different computers." As discussed earlier, the Mobile Programming System is written partly in ANSI FORTRAN. The portability of the entire system is somewhat dependent upon the ability to implement SIMCMP on the target computer. As a result of the variety of portable software which has been written, an ANSI FORTRAN based preprocessor/editor was written with the goal of achieving some degree of portibility.

IV. PREDITOR

PREDITOR (PReprocessor/EDITOR) was implemented in ANSI FORTRAN using the concepts of structured programming. This was considered to be a viable approach to machine independent software. PREDITOR is a preprocessor/editor which is capable of processing a small super-set of FORTRAN, called FORTRAN+, and allows changes to existing programs with a simplified text editor.

As an implementation language, FORTRAN is deficient in some areas. For example, ANSI FORTRAN does not provide a string data type. Some aspects of FORTRAN, such as the COMMON statement for representing global variables, have caused many man hours of work when a program change is necessary. FORTRAN+ attempts to solve some of the inadequacies of character manipulation inherent in FORTRAN and make the alteration of large software packages an easier task. In addition, the text editor section of PREDITOR includes four useful editing functions.

1. insert a line of code
2. delete a line or sequence of code
3. copy a predefined sequence of code
4. sequence the source file

The above functions are invoked using a number of "control toggles," a patch file containing desired changes, and a user's sequenced source file.

A. INDEPENDENT ASPECTS

Because of the availability of standard FORTRAN compilers, PREDITOR offers a highly portable package. Structured programming and machine-independent character representation allows the data structures and internal logic to be transparent to the user and target machine. The modular design has also allowed for the isolation of input/output processing. This results in a flexible system

for user reassignment of peripheral devices, particularly in the program transfer stage. Word size varies widely from machine to machine, and thus must be considered an important factor in portability. PREDITOR uses machine word size to its advantage. By packing applicable input data, memory requirements are minimized. This allows existing software to be preprocessed for use on target machines with different word lengths while minimizing machine storage requirements.

B. DEPENDENT ASPECTS

PREDITOR contains some machine dependencies. The data which PREDITOR must access during execution is packed according tc host machine word size. If PREDITCR is transferred to another machine, this data must be repacked acccrding to the new machine word size. PREDITOR itself can be used to process the appropriate statements and provide a copy of the reformatted data. The syntax tables which are vital to the compiler portion of PREDITOR are also machine dependent and can be reconstructed for the hcst machine.

Special characters which trigger PREDITOR functions may conflict with predefined uses of those characters in existing target machines. For example, the dollar sign is used as a ccmment indicator in the XDS Sigma 7. The reserved dollar character can easily be changed in any implementation.

C. METHCDS CF TRANSFER

Prior to transfer to a new environment, certain dependencies must be accounted for. Word size dependent syntax tables and packed output data must be replaced with tables and data generated for the target machine.

Input/output files presently reflect the file numbers of the IBM System 360. These file numbers may be redefined to meet user requirements or existing system file definitions. All areas which may need to be altered are clearly marked in the source listing of PREDITOR.

If a conflict involving the use of special characters

which trigger PREDITOR functions exists, PREDITOR must be altered. An alternate character must be defined and implemented in the system.

D. OPERATING PROCEDURES

PREDITOR is a one pass system. During the pass, the source program is scanned and all improperly formed statements are detected. A listing of the source program can be obtained and errors are listed by line number. Error messages take the form:

```
***** ERROR(x) NEAR LINE y
***** Brief explanation of error
```

The number x corresponds to the error number and y is the line number where the error occured. The line which immediately follows gives a brief explanation of the particular error message. See Appendix C for a complete list of error messages.

File numbers used in PREDITOR, along with the corressponding device types are listed in Appendix D. These I/O file numbers may be set prior to execution or during execution by using available control switches.

PREDITOR begins by reading a record from the input file. The PREDITOR processor translates the FORTRAN+ input text directly to FORTRAN statements, which are then compiled by the user's FORTRAN compiler. $$ Control switches are usually set at this time by the user. If not, control switches are set to default values. If the edit mode is not selected, the source file will be preprocessed as is. If editing is desired, GOFERS will merge the patch deck and the source file and process it normally. Figures 1 and 2 show typical patch deck and source file composition. When the nines cards are read, the operation is complete. The resulting merged file is then ready to be compiled from the temporary storage area.

```
$$ Control Switches

    (these consist of either control toggles
     or control switches and are used as desired.)


$$PATCH        (Required)

$$ Statements

    (These cards are used to delete or add code
     to the source file.)


99999999  (nines card required.)
```

Figure 1

A Typical Preditor Patch File

```
$$xx Statements (if desired)

    (These are used in conjunction with
     the $ COPY command)


$$ SOURCE      (Required)

$ Command Statements

    (These include $ OUTPUT, $ COPY and
     $ APPEND VERSE.)


(nines card required)                    99999999
```

Figure 2

A Typical Preditor Source File

E. $$ STATEMENTS

The $$ Statements are used to initialize PREDITCR and alter the predefined states of control toggles. $$ SOURCE marks the beginning of the source file. All statements following the $$ SOURCE are considered as part of the source file until a "nines card" (99999999 in columns 73-80) is detected.

The $$ PATCH incicates the beginning of the patch deck. The patch deck defines additions and deletions to the source file. All statements following the $$ PATCH are considered as part of the patch deck until a nines card (99999999 in columns 1-8) is detected. Patch deck cards are of two forms:

1. insert a card of code
   xxxxxxxx CODE BODY
2. delete a body of code
   xxxxxxxx*yyyyyyyy

The xxxxxxxx field is in columns 1 - 8 and indicates that the CODE BODY (columns 73-80) is to be inserted prior to a source file record of equal or greater sequence number. If a star (*) appears in column nine, the records to be deleted begin with sequence number xxxxxxxx and end with sequence number yyyyyyyy inclusive.

$$xx statements are used to define a portion of text which is to be copied into the executable source file. The dollar signs ($$) are placed in columns 1 and 2 with the xx field in columns 3 and 4. The xx field is any integer between 1 and 99 and represents all text which follows until another dollar sign is found in column 1. This text may be copied into the new source file with the call $ COPY xx. This allows the user to make changes to statements such as FORTRAN COMMON, which may be used frequently throughout a program. To facilitate the change, the user need only make the change in the definition stage as mentioned above. All $$xx statements and their associated text are physically placed directly before the $$ SOURCE card.

Other control switches take two forms: control toggles and control parameters. Toggles take on the values 1 and 0 (true or false), while parameters take on appropriate values.

Control switches are specified by typing a $$ in columns 1 and 2, and a switch name starting in column 3 (only the first character of the switch name is significant). The switch name is followed by an equal sign (=) and an integer value. Control switches are listed in Appendix B.

E. $ COMMAND STATEMENTS

The $ Command Statements are used freely within the source file after the $$ SOURCE card. These statements are part of the processor portion of PREDITOR. PREDITOR translates $ Command Statements to standard FORTRAN subroutine calls.

1. $ OUTPUT

The $ OUTPUT allows output of strings and integers. The use of the concatenation feature within the OUTPUT statement provides for a continuous output of strings and integers. A dollar sign ($) is placed in column 1 of the source file, followed by at least one blank, with the remainder of the statement being free format. The OUTPUT statement takes the following form:

```
$ OUTPUT(          )
```

All information to be written out is contained between the two parentheses. The statement must terminate with a right parenthesis prior to column 73.

a. Strings

Strings are delimited by the single quote ('). All characters which appear between two quotes are considered part of the string. The use of two adjacent quote marks inside the string results in the output of a single quote. Although the maximum length of text that one OUTPUT statement can process is 64 characters, the output of a longer text can be performed using the concatenation

feature in ccnjunctiqn with as many OUTPUT statements as are needed. Concatenation may be used to suppress dumping of the output buffer. This allows strings or integers which follow to be placed in the current buffer as shown in the following example.

```
$ OUTPUT('THIS IS A TEST '//)
$ OUTPUT(//'STRING')
```

The output would appear as THIS IS A TEST STRING.

b. Integers

Integers may be represented as variable names or constants. Four bases are available: binary, octal, decimal, or hexadecimal. A variable field width is also available with the option of blanks or zeros to fill out the field. To output an integer variable or constant, the statement must be of the form:

(x) nnnnnn : y

The value x in parentheses is optional and specifies a field width. If none is specified, a default width of 5 is used. If a negative value is given, the field is padded with blanks instead of zeros. The nnnnnn field is the variable or constant field, while the y represents the radix desired and is also optional, but defaults to base 10 if none is specified. All three components may be either integer values or variables with integer values. The negative concatenation option (/-/) can be used effectively in conjunction with the field width to produce a desired output. A /-/ will cause all leading blanks in a field width to be deleted. This allows integers to be left-justified when concatenated with strings or other numbers as shown in the following example.

```
$ OUTPUT('THE VALUE OF K IS '/-/(-10)K:10)
```

The output would appear as THE VALUE OF K IS 5 where K has the value 5.

2. $_COPY

The $COPY statement causes the text associated with the statement to be inserted at that point in the source

code. The statement must take the form:

$ COPY xx

The dollar sign ($) is placed in column 1, followed by at least one blank, with the remainder of the statement free format. The number xx is any integer between 1 and 99.

3. $ APPEND VERSE

The $ APPEND VERSE has the form:

$ APPEND VERSE

The dollar sign must again be placed in column 1, followed by at least one blank, with the remainder of the statement format free. This causes the vector VERSE, which contains the packed output data, to be copied into the new source file at this point. The associated FORTRAN equivalence statements will be built at this time.

V. SUMMARY

Within the areas discussed, each effort made to increase the portability of software has offered some distinct advantages. Both the Mobile Programming System and LIMP are macro prccessing systems which were developed for constructing machine independent software, but attack the problem with different concepts in mind. While the Mobile Programming System was designed to be transferred with the software, LIMP must meet this goal by processing programs at those installations which support the language WISP. The Mobile Programming System is capable of being bootstrapped on most computing machines to a level which best meets the needs of the software to be processed. LIMP, on the other hand, was designed to translate the software into cne of eight different languages. Although LIMP itself is bound to those systems which support WISP, the resulting software may be transferred to any machine which supports one of those eight languages.

The use of high-level language programming has alsc made a significant contribution to software portability. Standardization of programming languages and the attempt by many software manufacturers to produce a product which is easily understood and modified, has made transferability of software a much easier problem to solve. All methods discussed have experienced a degree of success. There is, however, another question which must be considered when the subject of portability arises. That is, what is the efficiency cf the final product. Because most systems designed for transferability are coded with the concept of structued programming in mind, critical routines can be singled cut and recoded in assembly language to increase efficiency. Although there are currently no comparative studies of efficiency of the macro processor systems

mentioned, Waite [Ref. 6] suggests that the "effort involved in this recoding is miniscule compared with that involved in hand-coding the entire processor." The success of this methodology is evident in the implementation of PL/M on the IBM System 360. With a minimum amount of alteration, efficiency was increased by approximately forty percent.

In the final analysis, the efficiency of the systems and methodologies used will be determined by the degree of development in those areas. The mobility associated with macro processors and high-level languages, however, has been demonstrated as a viable tool for implementation of programming languages on a variety of computing machines.

APPENDIX A: FORMAL DEFINITION OF PREDITOR

```
 1. <PROGRAM> ::= <STATEMENT LIST>

 2. <STATEMENT LIST> ::= <STATEMENT>
 3.                    | <STATEMENT LIST><STATEMENT>

 4. <STATEMENT> ::= <OUTPUT STATEMENT>
 5.                | <APPEND STATEMENT>
 6.                | <COPY STATEMENT>

 7. <OUTPUT STATEMENT> ::= <OUTPUT HEAD>  )

 8. <APPEND STATEMENT> ::= <APPEND><SUBJECT>

 9. <COPY STATEMENT> ::= <COPY><NUMBER>

10. <OUTPUT HEAD> ::= <OUTPUT>  (
11.                 | <OUTPUT HEAD><STRING>
12.                 | <OUTPUT HEAD><CONCAT>
13.                 | <OUTPUT HEAD><NCONCAT>
14.                 | <OUTPUT HEAD><CONOUT CALL>

15. <CONOUT CALL> ::= <CONOUT STATEMENT>
16.                 | <CONOUT CALL><RADIX>

17. <CONOUT STATEMENT> ::= <IDENTIFIER>
18.                      | <NUMBER>
19.                      | <FIELD WIDTH><IDENTIFIER>
20.                      | <FIELD WIDTH><NUMBER>

21. <FIELD WIDTH> ::= ( <NUMBER> )

22. <RADIX> ::= <RADIX HEAD><IDENTIFIER>
23.           | <RADIX HEAD><NUMBER>

24. <RADIX HEAD> ::=  :

25. <COPY> ::=  COPY

26. <OUTPUT> ::=  OUTPUT

27. <APPEND> ::=  APPEND

28. <SUBJECT> ::=  VERSE
```

APPENDIX B: PREDITOR CONTROL TOGGLES

Switch Name	Use	Default
$$APATCH=n	The patch deck input file number = n	1
$$BSEQUENCE=n	Begin:sequence numbering at n	25
$$DELTA=n	Sequence number increment = n	25
$$EMARGIN=n	Set the output line width to n characters.	120
$$HWDSIZE=n	Host machine wordsize = n	32
$$INPUT=n	Switch to file n for subsequent input ,(see file numbering).	1
$$MERGE	Edit mode: T or F	0
$$OUTPUT=n	Write subsequent output lines to file n	1
$$PRINT	Output lines to the printer: T or F	1
$$RMARGIN=n	Logical record size = n	80
$$SEQUENCE	Resequence the source file: T or F	0
$$WORDSIZE=n	Target machine wordsize = n	32

NOTE: All input files are maximum of 80 characters, and the output files cannot exceed 120 characters.

APPENDIX C: ERROR MESSAGES

Error Number	Message
1	Illegal symbol pair. The symbols printed cannot appear in a valid statement. This error may have been caused by a previous error in the program.
2	Parse stack overflow. Simplify the statement or declare a larger parse stack and recompile.
3	Table overflow. Error may be result of too many variables in use at one time. Either simplify the OUTPUT statement or increase the varc size and recompile.
4	Improperly formed statement. The statement above is improperly formed. Check formal BNF for definition and usage.
5	Incorrect format for concatenation used. Proper format is: // : Append to the current output buffer. /-/ : Append to the current output buffer and delete the leading blanks.
6	String length exceeds allowable limits. Use shorter string length combined with concatenation.
7	Variable exceeds FORTRAN limits. Variable listed below is greater than six characters in length. Rename the variable.
8	A $$CONTROL statement is improperly formed. Check all CONTROL statements to insure proper format. $$ SOURCE $$ PATCH $$CONTROL TOGGLE

APPENDIX D: FILE DEFINITIONS

INPUT				OUTPUT		
NUMBER	DEVICE	UNIT	*	NUMBER	DEVICE	UNIT
1	CARDRDR	5	*	1	PRINTER	6
2	TELTYPE	4	*	2	TELTYPE	4
3	PAPTAPE	6	*	3	PUNCH	7
4	MAGTAPE	7	*	4	MAGTAPE	8
5	DSKFILE	8	*	5	DSKFILE	9
6	DSKFILE	9	*	6	DSKFILE	10
7	AVAILABLE	10	*	7	PAPTAPE	11

NOTE: Unit numbers may be altered to fit user requirements by accessing subroutines RDTEXT and WRITEL. These are the only references to these unit numbers.

```
C     ************************************************
C     * THIS IS A SAMPLE OF A SOURCE FILE PRIOR TO BEING   *
C     * PROCESSED BY PREDITOR                              *
C     ************************************************
C
$$10  INTEGER IBUFF(80),OBUFF(120),IBP,OBP,OTRAN(64),MARGIN                    00000005
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN                          00000055
$$20  INTEGER CONTRL(64),IFILE,OFILE                                           00000105
      COMMON /CONTRL/ CONTRL,IFILE,OFILE                                       00000155
$$30  INTEGER VERSE(500)                                                       00000205
      COMMON /TEXT/ VERSE                                                      00000255
$$    SOURCE                                                                   00000305
$     COPY 10                                                                  00000320
$     COPY 20                                                                  00000325
$     COPY 30                                                                  00000335
      MARGIN = 120                                                             00000355
      CONTRL(20) = 1                                                           00000405
      CONTRL(26) = 1                                                           00000455
      IFILE = CONTRL(20)                                                       00000505
      OFILE = CONTRL(26)                                                       00000555
$     OUTPUT ('THIS IS A TEST OF PREDITOR')                                    00000605
      I = 4                                                                    00000655
      J = 4                                                                    00000705
      K = I + J                                                                00000755
$     OUTPUT ('THE VALUE OF K IS '/-/ (-5) K : 10)                             00000805
      STOP                                                                     00000855
      END                                                                      00000905
```

```
      SUBROUTINE WRITEL(NSPACE)                                          00000955
$     COPY 10                                                            00001005
$     COPY 20                                                            00001055
      IF(OBP .EQ. 0) GO TO 999                                           00001105
      NBLANK = 1                                                         00001155
      DO 5 I=1,OBP                                                       00001205
          J = OBUFF(I)                                                   00001255
          IF(J .NE. 1) NBLANK = I                                        00001305
          OBUFF(I) = OTRAN(J)                                            00001355
    5 CONTINUE                                                           00001405
      OBP = IMIN(120,NBLANK)                                             00001455
      GO TO (10),OFILE                                                   00001505
   10 WRITE(6,1002) (OBUFF(I),I=1,OBP)                                   00001555
   11 IF(NSPACE .EQ. 0) GO TO 20                                         00001605
      DO 15 I=1,NSPACE                                                   00001655
          WRITE(6,1000)                                                  00001705
   15 CONTINUE                                                           00001755
   20 OBP = 0                                                            00001805
  999 RETURN                                                             00001655
 1001 FORMAT(1H )                                                        00001905
 1002 FORMAT(1H ,120A1)                                                  00001955
      END                                                                00002005
```



```
      SUBROUTINE PAD(LC,CHR,I)                                           00002055
      INTEGER LC,CHR,I,T(20)                                             00002105
      J = IMIN(I,20)                                                     00002155
      DO 10 K=1,J                                                        00002205
          T(K) = CHR                                                     00002255
   10 CONTINUE                                                           00002305
      CALL FORM(LC,T,1,J,20,.FALSE.)                                     00002355
      RETURN                                                             00002405
      END                                                                00002455
```

```
      SUBROUTINE CONOUT(LC,K,N,BASE)                                    00002505
      INTEGER K,N,BASE,T(20),LC                                         00002555
$     COPY 10                                                           00002605
      LOGICAL ZSUP,AFLAG                                                00002655
      NP = N                                                            00002705
      ZSUP = K .LT. 0                                                   00002755
      KP = IMIN(IABS(K),19)                                             00002805
      DO 10 I=1,KP                                                      00002855
         T(I) = 1                                                       00002905
   10 CONTINUE                                                          00002955
      IP = KP + 1                                                       00003005
      DO 20 I=1,KP                                                      00003055
         T(IP-I) = MOD(NP,BASE) + 2                                     00003105
         NP = NP/BASE                                                   00003155
         IF(ZSUP .AND. (NP .EQ. 0)) GO TO 30                            00003205
   20 CONTINUE                                                          00003255
      AFLAG = .FALSE.                                                   00003305
   30 IF(.NOT. AFLAG) GO TO 35                                          00003355
   35 IF(BASE .EQ. 8) GO TO 40                                          00003405
      IF(BASE .EQ. 2) GO TO 45                                          00003455
      IF(BASE .NE. 16) GO TO 50                                         00003505
      KP = KP + 1                                                       00003555
      T(KP) = 19                                                        00003605
      GO TO 50                                                          00003655
   40 KP = KP + 1                                                       00003705
      T(KP) = 28                                                        00003755
      GO TO 50                                                          00003805
   45 KP = KP + 1                                                       00003855
      T(KP) = 13                                                        00003905
   50 CALL FORM(LC,T,1,KP,20,.FALSE.)                                   00003955
      RETURN                                                            00004005
      END                                                               00004055
```

```
      SUBROUTINE FORM(LC,CHARS,START,FINISH,LENGTH,UNPAC)                00004105
$     COPY 10                                                            00004155
$     COPY 20                                                            00004205
      INTEGER CHARS(LENGTH),LC,START,FINISH,DUMLEN                       00004255
      LOGICAL UNPAC                                                      00004305
      DUMLEN = LENGTH                                                    00004355
      J = START                                                          00004405
      I = LC + 1                                                         00004455
      GO TO (100,200,300),I                                              00004505
  100 CALL WRITEL(0)                                                     00004555
  200 IF(UNPAC) GO TO 250                                                00004605
      IF(J .GT. FINISH) GO TO 999                                        00004655
      OBP = OBP + 1                                                      00004705
      OBUFF(OBP) = CHARS(J)                                              00004755
      J = J + 1                                                          00004805
      IF (OBP .GE. MARGIN) GO TO 100                                     00004855
      GO TO 200                                                          00004905
  250 N = CONTRL(19)/6                                                   00004955
  260 IF(N .LE. DUMLEN) GO TO 270                                        00005005
      N = MOD(DUMLEN,N)                                                  00005055
  270 K = OBP + N                                                        00005105
      IF (K .GT. MARGIN) CALL WRITEL(0)                                  00005155
      OBP = OBP + N                                                      00005205
      CALL UNPACK(CHARS(J),N)                                            00005255
      DUMLEN = DUMLEN - N                                                00005305
      J = J + 1                                                          00005355
      IF(J .GT. FINISH) GO TO 999                                        00005405
      GO TO 260                                                          00005455
  300 IF(J .GT. FINISH) GO TO 999                                        00005505
      IF(CHARS(J) .NE. 1) GO TO 200                                      00005555
      J = J + 1                                                          00005605
      GO TO 300                                                          00005655
  999 RETURN                                                             00005705
      END                                                                00005755
```

```
      SUBROUTINE UNPACK(WORD,I)                                          00005805
$     COPY 10                                                            00005855
$     COPY 20                                                            00005905
      INTEGER SHL,SHR,SHLCNT,WORD,I                                      00005955
      SHLCNT = 0                                                         00006005
      DO 100 J=1,I                                                       00006055
          K = SHL(WORD,CONTRL(19)-6*J)                                   00006105
          IF(K .GE. 0) GO TO 80                                          00006155
   50     IF((K .GE. 0).OR.(SHLCNT.GE.6)) GO TO 60                       00006205
          SHLCNT = SHLCNT + 1                                            00006255
          K = SHL(WORD,CONTRL(19)+SHLCNT-6*J)                            00006305
          GO TO 50                                                       00006355
   60     K = SHR(K,CONTRL(19)+SHLCNT-6)                                 00006405
          DO 70 L=1,SHLCNT                                               00006455
              K = K + 2**(6-L)                                           00006505
   70     CONTINUE                                                       00006555
          SHLCNT = 0                                                     00006605
          GO TO 90                                                       00006655
   80     K = SHR(K,CONTRL(19)-6)                                        00006705
   90     OBUFF(OBP-J+1) = K                                             00006755
  100 CONTINUE                                                           00006805
      RETURN                                                             00006855
      END                                                                00006905


      INTEGER FUNCTION SHL(I,J)                                          00006955
      SHL = I*(2**J)                                                     00007005
      RETURN                                                             00007055
      END                                                                00007105


      INTEGER FUNCTION SHR(I,J)                                          00007155
      SHR = I/(2**J)                                                     00007205
      RETURN                                                             00007255
      END                                                                00007305


      FUNCTION IMIN(I,J)                                                 00007355
      IF(I .LT. J) GO TO 10                                              00007405
      IMIN = J                                                           00007455
      GO TO 999                                                          00007505
   10 IMIN = I                                                           00007555
  999 RETURN                                                             00007605
      END                                                                00007655
```

```
      BLOCK DATA                                                                00007705
$     COPY 10                                                                   00007755
      DATA IBUFF/80*0/,OBUFF/120*0/,IBP/81/,OBP/0/                              00007805
      DATA OTRAN/1H ,1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,                   00007855
     1          1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,1HK,                   00007905
     2          1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,                   00007955
     3          1HW,1HX,1HY,1HZ,                                               00008005
     4          1H$,1H=,1H.,1H/,1H(,1H),1H+,1H-,1H',1H*,1H,,                   00008055
     5          1H<,1H>,1H:,1H;,12*0/                                          00008105
$     COPY 20                                                                   00008155
      DATA CONTRL/64*0/,IFILE/0/,OFILE/0/                                       00008205
$     COPY 30                                                                   00008255
$     APPEND VERSE                                                              00008305
      END                                                                       00008355
                                                                                99999999
```

```
C      ************************************************************
C      * THE FOLLOWING IS A PATCH DECK TO BE MERGED WITH          *
C      * THE SOURCE FILE.  THE RESULTING PROGRAM CAN BE RUN       *
C      * ON A MACHINE WITH TWENTY BITS OR MORE.                   *
C      ************************************************************
C
$$M=1
$$I=6
$$W=20
$$O=3
$$S=1
$$   PATCH
00000400       CONTRL(19) = 20
00000645$      OUTPUT ('THIS IS AN ADDED CARD')
00000655*00000705
00000720       I = 2
00000750       J = 2
00001705*00001705
00001750           WRITE(6,1001)
99999999
```

```
C     ************************************************
CC    * THIS PROGRAM HAS BEEN PREPROCESSED BY PREDITOR.    *
CC    * THE RESULT IS A RESEQUENCED FILE WHICH CAN BE RUN  *
CC    * ON ANY MACHINE WHERE THERE IS A FORTRAN COMPILER   *
CC    ************************************************
CC
C     COPY 10                                                          00000000
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,OTRAN(64),MARGIN            00000005
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN                  00000055
C     COPY 20                                                          00000050
      INTEGER CONTRL(64),IFILE,OFILE                                   00000105
      COMMON /CONTRL/ CONTRL,IFILE,OFILE                               00000155
C     COPY 30                                                          00000100
      INTEGER VERSE(500)                                               00000205
      COMMON /TEXT/ VERSE                                              00000255
      MARGIN = 120                                                     00000150
      CONTRL(19) = 20                                                  00000200
      CONTRL(20) = 1                                                   00000250
      CONTRL(26) = 1                                                   00000300
      IFILE = CONTRL(20)                                               00000350
      OFILE = CONTRL(26)                                               00000400
C     OUTPUT ('THIS IS A TEST OF PREDITOR')                            00000450
      CALL FORM(0,VERSE,1,9,26,.TRUE.)
      CALL WRITEL(0)
C     OUTPUT ('THIS IS AN ADDED CARD')                                 00000500
      CALL FORM(0,VERSE,10,16,21,.TRUE.)
      CALL WRITEL(0)
      I = 2                                                            00000550
      J = 2                                                            00000600
      K = I + J                                                        00000650
C     OUTPUT ('THE VALUE OF K IS '/-/ (-5) K : 10)                     00000700
      CALL FORM(0,VERSE,17,22,18,.TRUE.)
      CALL CONOUT(2,-5    ,K     ,10    )
      CALL WRITEL(0)
      STOP                                                             00000750
      END                                                              00000800
```

```
      SUBROUTINE WRITEL(NSPACE)                                          00000850
C     COPY 10                                                            00000900
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,OTRAN(64),MARGIN              00000005
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN                    00000055
C     COPY 20                                                            00000950
      INTEGER CONTRL(64),IFILE,OFILE                                     00000105
      COMMON /CONTRL/ CONTRL,IFILE,OFILE                                 00000155
      IF(OBP .EQ. 0) GO TO 999                                           00001000
      NBLANK = 1                                                         00001050
      DO 5 I=1,OBP                                                       00001100
          J = OBUFF(I)                                                   00001150
          IF(J .NE. 1) NBLANK = I                                        00001200
          OBUFF(I) = OTRAN(J)                                            00001250
    5 CONTINUE                                                           00001300
      OBP = IMIN(120,NBLANK)                                             00001350
      GO TO (10),OFILE                                                   00001400
   10 WRITE(6,1002) (OBUFF(I),I=1,OBP)                                   00001450
   11 IF(NSPACE .EQ. 0) GO TO 20                                         00001500
      DO 15 I=1,NSPACE                                                   00001550
          WRITE(6,1001)                                                  00001600
   15 CONTINUE                                                           00001650
   20 OBP = 0                                                            00001700
  999 RETURN                                                             00001750
 1001 FORMAT(1H )                                                        00001800
 1002 FORMAT(1H ,120A1)                                                  00001850
      END                                                                00001900


      SUBROUTINE PAD(LC,CHR,I)                                           00001950
      INTEGER LC,CHR,I,T(20)                                             00002000
      J = IMIN(I,20)                                                     00002050
      DO 10 K=1,J                                                        00002100
          T(K) = CHR                                                     00002150
   10 CONTINUE                                                           00002200
      CALL FORM(LC,T,1,J,20,.FALSE.)                                     00002250
      RETURN                                                             00002300
      END                                                                00002350
```

```
      SUBROUTINE CONOUT(LC,K,N,BASE)                                    00002400
      INTEGER K,N,BASE,T(20),LC                                         00002450
C     COPY 10                                                           00002500
      INTEGER IBUFF(80),OBUFF(120),IBP,CBP,OTRAN(64),MARGIN             00000005
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN                   00000055
      LOGICAL ZSUP,AFLAG                                                00002550
      NP = N                                                            00002600
      ZSUP = K .LT. 0                                                   00002650
      KP = IMIN(IABS(K),19)                                             00002700
      DO 10 I=1,KP                                                      00002750
         T(I) = 1                                                       00002800
   10 CONTINUE                                                          00002850
      IP = KP + 1                                                       00002900
      DO 20 I=1,KP                                                      00002950
         T(IP-I) = MOD(NP,BASE) + 2                                     00003000
         NP = NP/BASE                                                   00003050
         IF(ZSUP .AND. (NP .EQ. 0)) GO TO 30                            00003100
   20 CONTINUE                                                          00003150
      AFLAG = .FALSE.                                                   00003200
   30 IF(.NOT. AFLAG) GO TO 35                                          00003250
   35 IF(BASE .EQ. 8) GO TO 40                                          00003300
      IF(BASE .EQ. 2) GO TO 45                                          00003350
      IF(BASE .NE. 16) GO TO 50                                         00003400
      KP = KP + 1                                                       00003450
      T(KP) = 19                                                        00003500
      GO TO 50                                                          00003550
   40 KP = KP + 1                                                       00003600
      T(KP) = 28                                                        00003650
      GO TO 50                                                          00003700
   45 KP = KP + 1                                                       00003750
      T(KP) = 13                                                        00003800
   50 CALL FORM(LC,T,1,KP,20,.FALSE.)                                   00003850
      RETURN                                                            00003900
      END                                                               00003950
```

```
      SUBROUTINE FORM(LC,CHARS,START,FINISH,LENGTH,UNPAC)                00004000
C     COPY 10                                                            00004050
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,OTRAN(64),MARGIN              00000005
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN                    00000055
C     COPY 20                                                            00004100
      INTEGER CONTRL(64),IFILE,OFILE                                     00000105
      COMMON /CONTRL/ CONTRL,IFILE,OFILE                                 00000155
      INTEGER CHARS(LENGTH),LC,START,FINISH,DUMLEN                       00004150
      LOGICAL UNPAC                                                      00004200
      DUMLEN = LENGTH                                                    00004250
      J = START                                                          00004300
      I = LC + 1                                                         00004350
      GO TO (100,200,300),I                                              00004400
  100 CALL WRITEL(0)                                                     00004450
  200 IF(UNPAC) GO TO 250                                                00004500
      IF(J .GT. FINISH) GO TO 999                                        00004550
      OBP = OBP + 1                                                      00004600
      OBUFF(OBP) = CHARS(J)                                              00004650
      J = J + 1                                                          00004700
      IF (OBP .GE. MARGIN) GO TO 100                                     00004750
      GO TO 200                                                          00004800
  250 N = CONTRL(19)/6                                                   00004850
  260 IF(N .LE. DUMLEN) GO TO 270                                        00004900
      N = MOD(DUMLEN,N)                                                  00004950
  270 K = OBP + N                                                        00005000
      IF (K .GT. MARGIN) CALL WRITEL(0)                                  00005050
      OBP = OBP + N                                                      00005100
      CALL UNPACK(CHARS(J),N)                                            00005150
      DUMLEN = DUMLEN - N                                                00005200
      J = J + 1                                                          00005250
      IF(J .GT. FINISH) GO TO 999                                        00005300
      GO TO 260                                                          00005350
  300 IF(J .GT. FINISH) GO TO 999                                        00005400
      IF(CHARS(J) .NE. 1) GO TO 200                                      00005450
      J = J + 1                                                          00005500
      GO TO 300                                                          00005550
  999 RETURN                                                             00005600
      END                                                                00005650
```

```
      SUBROUTINE UNPACK(WORD,I)                                          00005700
C     COPY 10                                                            00005750
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,OTRAN(64),MARGIN              00000005
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN                    00000055
C     COPY 20                                                            00005800
      INTEGER CONTRL(64),IFILE,OFILE                                     00000105
      COMMON /CONTRL/ CONTRL,IFILE,OFILE                                 00000155
      INTEGER SHL,SHR,SHLCNT,WORD,I                                      00005850
      SHLCNT = 0                                                         00005900
      DO 100 J=1,I                                                       00005950
          K = SHL(WORD,CONTRL(19)-6*J)                                   00006000
          IF(K .GE. 0) GO TO 80                                          00006050
   50     IF((K .GE. 0).OR.(SHLCNT.GE.6)) GO TO 60                       00006100
          SHLCNT = SHLCNT + 1                                            00006150
          K = SHL(WORD,CONTRL(19)+SHLCNT-6*J)                            00006200
          GO TO 50                                                       00006250
   60     K = SHR(K,CONTRL(19)+SHLCNT-6)                                 00006300
          DO 70 L=1,SHLCNT                                               00006350
              K = K + 2**(6-L)                                           00006400
   70     CONTINUE                                                       00006450
          SHLCNT = 0                                                     00006500
          GO TO 90                                                       00006550
   80     K = SHR(K,CONTRL(19)-6)                                        00006600
   90     OBUFF(OBP-J+1) = K                                             00006650
  100 CONTINUE                                                           00006700
      RETURN                                                             00006750
      END                                                                00006800

      INTEGER FUNCTION SHL(I,J)                                          00006850
      SHL = I*(2**J)                                                     00006900
      RETURN                                                             00006950
      END                                                                00007000

      INTEGER FUNCTION SHR(I,J)                                          00007050
      SHR = I/(2**J)                                                     00007100
      RETURN                                                             00007150
      END                                                                00007200

      FUNCTION IMIN(I,J)                                                 00007250
      IF(I .LT. J) GO TO 10                                              00007300
      IMIN = J                                                           00007350
      GO TO 999                                                          00007400
   10 IMIN = I                                                           00007450
  999 RETURN                                                             00007500
      END                                                                00007550
```

```
      BLOCK DATA                                                        00007600
C     COPY 10                                                           00007650
      INTEGER IBUFF(80),OBUFF(120),IBP,CBP,OTRAN(64),MARGIN             00000005
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,OTRAN,MARGIN                   00000055
      DATA IBUFF/80*0/,OBUFF/120*0/,IBP/81/,OBP/0/                      00007700
      DATA OTRAN/1H ,1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,           00007750
     1           1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,1HK,          00007800
     2           1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,          00007850
     3           1HW,1HX,1HY,1HZ,                                      00007900
     4           1H$,1H=,1H.,1H/,1H(,1H),1H+,1H-,1H',1H*,1H,,          00007950
     5           1H<,1H>,1H:,1H;,12*0/                                 00008000
C     COPY 20                                                           00008050
      INTEGER CONTRL(64),IFILE,OFILE                                    00000105
      COMMON /CONTRL/ CONTRL,IFILE,OFILE                                00000155
      DATA CONTRL/64*0/,IFILE/0/,OFILE/0/                               00008100
C     COPY 30                                                           00008150
      INTEGER VERSE(500)                                                00000205
      COMMON /TEXT/ VERSE                                               00000255
C     APPEND VERSE                                                      00008200
      DATA VERSE/
     *128212,122964,122956,6096,124865,107585,112464,62751,1693,128212,
     *122964,122956,102476,62416,61518,51023,128208,6220,96272,5777,
     *5505,83841,
     *478*0/
      END                                                               00008250
```

ERROR MESSAGES

ERROR NUMBER	MESSAGE
1	ILLEGAL SYMBOL PAIR. THE SYMBOLS PRINTED CANNOT APPEAR IN A VALID STATEMENT. THIS ERROR MAY HAVE BEEN CAUSED BY A PREVIOUS ERROR IN THE PROGRAM.
2	PARSE STACK OVERFLOW. SIMPLIFY THE STATEMENT OR DECLARE A LARGER PARSE STACK AND RECOMPILE.
3	TABLE OVERFLOW. ERROR MAY BE RESULT OF TOO MANY VARIABLES IN USE AT ONE TIME. EITHER SIMPLIFY THE OUTPUT STATEMENT OR INCREASE THE VARC SIZE AND RECOMPILE.
4	IMPROPERLY FORMED STATEMENT. THE STATEMENT ABOVE IS IMPROPERLY FORMED. CHECK FORMAL BNF FOR LANGUAGE DEFINITION AND USAGE.
5	INCORRECT FORMAT FOR CONCATENATION USED. PROPER FORMAT IS: // :APPEND TO THE CURRENT OUTPUT BUFFER /-/ :APPEND TO THE CURRENT OUTPUT BUFFER DELETING THE LEADING BLANKS
6	STRING LENGTH EXCEEDS ALLOWABLE LIMITS. USE SHORTER STRING LENGTH COMBINED WITH CONCATENATION.
7	VARIABLE EXCEEDS FORTRAN LIMITS. VARIABLE LISTED BELOW IS GREATER THAN SIX CHARACTERS IN LENGTH. RENAME THE VARIABLE.
8	A $$CONTROL STATEMENT IS IMPROPERLY FORMED. CHECK ALL CONTROL STATEMENTS TO INSURE PROPER FORMAT. $$ SOURCE $$CONTROL TOGGLE

****

FILE DEFINITIONS

INPUT NUMBER	INPUT DEVICE	INPUT UNIT	*	OUTPUT NUMBER	OUTPUT DEVICE	OUTPUT UNIT
1	CARDRDR	5	*	1	PRINTER	6
2	TELTYPE	4	*	2	TELTYPE	4
3	PAPTAPE	6	*	3	PUNCH	7
4	MAGTAPE	7	M	4	4 GTAPE	8
5	DSKFILE	8	*	5	DSKFILE	9
6	DSKFILE	9	*	6	DSKFILE	10
7	AVAILABLE	10	*	7	PAPTAPE	11

ALL INPUT RECORDS ARE 80 CHARACTERS OR LESS. ALL OUTPUT RECORDS ARE 120 CHARACTERS OR LESS. UNIT NUMBERS MAY BE ALTERED TO FIT USER REQUIREMENTS BY ACCESSING SUBROUTINES RDTEXT AND WRITEL. THESE ARE THE ONLY REFERENCES TO THESE UNIT NUMBERS.

****

INPUT RECORDS ARE TRANSLATED TO INTERNAL CODE FOR PROGRAM USE.

** INTERNAL CODE CONVERSION **

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	1	1
2	3	4	5	6	7	8	9	0	1

A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1	1	1	1	1	1	1	1	2	2	2	2	2	2	2	2	2	3	3	3	3	3	3	3	3	3
2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7	8	9	0	1	2	3	4	5	6	7

$	=	.	/	(	)	+	-	'	*	,	<	>	:	;
3	3	4	4	4	4	4	4	4	4	4	4	5	5	5
8	9	0	1	2	3	4	5	6	7	8	9	0	1	2

BLANKS HAVE INTERNAL CODE OF 1

```
      INTEGER MAXPAC,WDSIZE,LINENB
      COMMON /INIT/ MAXPAC,WDSIZE,LINENB
      INTEGER VCOPY(37),COMLEN(37),COMVEC(500),CPTR,
     1SOURCE(6),PCNT,COMFIN(37)
      LOGICAL ERFLAG,SORCE
      COMMON /COPYS/ VCOPY,COMLEN,COMVEC,CPTR,SOURCE,PCNT,
     1ERFLAG,SORCE,COMFIN
      INTEGER VERLEN,START,FINISH,LENGTH,PAKCNT,VERSE(500),
     1CC
      COMMON /FORMS/ VERLEN,START,FINISH,LENGTH,PAKCNT,
     1VERSE,CC
      INTEGER RADIX(8),VARB(8),FW(8)
      COMMON /CNOUT/ RADIX,VARB,FW
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),
     1OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,
     1COMPIL
      INTEGER ACCLEN,ACCUM(99),NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR(75),VARTOP,VARC(256),MVAR
      COMMON /SCANN/ACCLEN,ACCUM,NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR,VARTOP,VARC,MVAR
      LOGICAL AFLAG,TOGSET
      COMMON /FLAGS/ AFLAG,TOGSET
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER PABUFF(80),SOBUFF(80),SONUM(8),PANUM(8),PASNUM(8),MERGE
      LOGICAL INSERT,FAULT
      COMMON /MERGES/ PABUFF,SOBUFF,SONUM,PANUM,PASNUM,MERGE,INSERT,
     1FAULT
      INTEGER EQUIV(10),EQPTR,NUMCTR,NUMLN,MAXLN,OBPCK
      COMMON /EQUIVA/ EQUIV,EQPTR,NUMCTR,NUMLN,MAXLN,OBPCK
      INTEGER TEXT (138)
      COMMON /TEXTS/ TEXT
      INTEGER SEQN
      LOGICAL SEQ
      COMMON /SEQNS/ SEQN,SEQ
```

```
      INTEGER GNC
C
CCC   THE FOLLOWING CONTROLS ARE DEFINED AS FOLLOWS. THEY ARE SET
CCC   BY THE USER TO PROVIDE THE OPTIONS HE DESIRES.
CCC   OPTION                          CONTROL NUMBER
CCC
CCC   PATCH INPUT = I                 (12)
CCC   SEQUENCE NUMBER START = I       (13)
CCC   SEQUENCE NUM INCREMENT = I      (15)
CCC   PRINT MARGIN = I                (16)
CCC   HOST WDSIZE = I                 (19)
CCC   INPUT FILE = I                  (20)
CCC   LEFT MARGIN = I                 (23)
CCC   MERGE = T OR F                  (24)
CCC   OUTPUT FILE = I                 (26)
CCC   PRINT = T OR F                  (27)
CCC   RIGHT MARGIN = I                (29)
CCC   SEQUENCE = T OR F               (30)
CCC   TRIGGER = I                     (31)
CCC   TARGET WORD SIZE = I            (34)
C
      CONTRL(12) = 1
      CONTRL(13) = 0
      CONTRL(15) = 50
      CONTRL(16) = 120
      CONTRL(19) = 32
      CONTRL(20) = 1
      CONTRL(23) = 1
      CONTRL(24) = 0
      CONTRL(26) = 1
      CONTRL(27) = 1
      CONTRL(29) = 80
      CONTRL(30) = 0
      CONTRL(31) = 38
      CONTRL(34) = 32
      AFLAG = .FALSE.
      DO 1 I=1,5
         PRMASK(I) = 2**(I*8-8) - 1
    1 CONTINUE
      DO 2 I=1,256
         ITRAN(I) = 1
    2 CONTINUE
      DO 5 I=53,64
         OTRAN(I) = OTRAN(1)
    5 CONTINUE
      DO 10 I=1,52
         J = OTRAN(I)
         J = ICON(J)
```

```
      ITRAN(J) = I
   10 CONTINUE
      IF(CONTRL(27) .EQ. 0) GO TO 14
      OFILE = 1
      MARGIN = CONTRL(29)
      CALL FORM(0,TEXT ,1,9,42,.TRUE.)
      CALL WRITEL(1)
   14 IBP = 81
   15 I = GNC(0)
   16 J = 0
      IF (IBUFF(1).EQ.999) GO TO 999
      IF (IBUFF(1) .NE. CONTRL(31)) GO TO 13
      IF (IBUFF(2) .NE. CONTRL(31)) GO TO 30
      IF((IBUFF(3) .LE. 11) .AND. (IBUFF(3) .GE. 2)) GO TO 35
      CALL SETTOG
      WDSIZE = CONTRL(34)
      MAXPAC = WDSIZE/6
      GO TO 14
   35 CALL COPY
      GO TO 16
   30 J = 38
      IBUFF(1) = 14
   13 IF (CONTRL(30) .EQ. 0) GO TO 18
      CALL SEQDK
   18 IF(CONTRL(27) .EQ. 0) GO TO 17
      MARGIN = CONTRL(16)
      LINENB = LINENB + 1
      CALL CONOUT(0,5,LINENB,10)
      CALL PAD(1,1,3)
      CALL FORM(1,IBUFF,1,80,80,.FALSE.)
      OFILE = 1
      CALL WRITEL(0)
      MARGIN = CONTRL(29)
   17 IF (CONTRL(26) .EQ. 1) GO TO 20
      OFILE = CONTRL(26)
      CALL FORM(0,IBUFF,1,80,80,.FALSE.)
   20 IF (J .NE. CONTRL(31)) GO TO 14
      DO 25 I=1,3
         PSTACK(I) = 0
   25 CONTINUE
      PSTACK(4) = EOFILE
      SP = 4
      CALL SCAN
      CC = 0
      CALL CLOOP
      GO TO 16
  999 STOP
      END
```

```
      INTEGER FUNCTION GNC(IQ)
C
CCCCC ***************************************************
CCCCC * GNC GETS THE NEXT CHARACTER FROM THE INPUT STREAM. *
CCCCC * IF THE EDIT MODE IS SELCTED, GNC DECIDES WHICH     *
CCCCC * FILE TO READ FROM AND PASSES ON THE PROPER BUFFER. *
CCCCC ***************************************************
C
      INTEGER MAXPAC,WDSIZE,LINENB
      COMMON /INIT/ MAXPAC,WDSIZE,LINENB
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER PABUFF(80),SOBUFF(80),SCNUM(8),PANUM(8),PASNUM(8),MERGE
      LOGICAL INSERT,FAULT
      COMMON /MERGES/ PABUFF,SOBUFF,SCNUM,PANUM,PASNUM,MERGE,INSERT,
     1FAULT
      LOGICAL AFLAG,TOGSET
      COMMON /FLAGS/ AFLAG,TOGSET
      LOGICAL NINES,MERGIT
      IF(IBP .LE. 72) GO TO 999
      IF(.NOT.TOGSET) GO TO 340
      IF(CONTRL(24) .EQ. 0) GO TO 350
      IF(.NOT. INSERT) GO TO 100
      IFILE = CONTRL(12)
      CALL RDTEXT(3)
      CALL SEQNUM(1)
      IF(NINES(1)) CONTRL(24) = 0
  100 IF(INSERT .AND. FAULT) GO TO 200
      FAULT = .TRUE.
      IFILE = CONTRL(20)
      CALL RDTEXT(2)
      CALL SEQNUM(3)
      IF(.NOT. NINES(2)) GO TO 200
      IBUFF(1) = 999
      RETURN
  200 IF(.NOT. MERGIT(PANUM,SCNUM)) GO TO 300
      CALL MERG
      GO TO 400
  300 DO 310 I=1,80
          IBUFF(I) = SOBUFF(I)
  310 CONTINUE
      INSERT = .FALSE.
      GO TO 400
  340 IFILE = CONTRL(12)
      GO TO 351
  350 IFILE = CONTRL(20)
```

```
351 CALL RDTEXT(1)
    CALL SEQNUM(4)
    IF(.NOT. NINES(2)) GO TO 400
    IBUFF(1) = 999
    RETURN
400 IBP = 2
    GNC = 0
    RETURN
999 GNC = IBUFF(IBP)
    IBP = IBP + 1
    RETURN
    END
```

```
      SUBROUTINE RDTEXT(L)
C     ****************************************************
C     * RDTEXT READS FROM THE PROPER INPUT FILE AND OPAC
C     * RDTEXT READS FROM THE PROPER INPOT FILE AN& PLACES *
C     * IT IN THE APPROPRIATE BUFFER.                      *
C     ****************************************************
C
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER PABUFF(80),SOBUFF(80),SONUM(8),PANUM(8),PASNUM(8),MERGE
      LOGICAL INSERT,FAULT
      COMMON /MERGES/ PABUFF,SOBUFF,SONUM,PANUM,PASNUM,MERGE,INSERT,
     1FAULT
      INTEGER BUFFER(80)
      GO TO (10,50,60,70,80,90,100),IFILE
   10 READ(5,1000) BUFFER
      GO TO 115
   50 READ(4,1000) BUFFER
      GO TO 115
   60 READ(6,1000) BUFFER
      GO TO 115
   70 READ(7,1000) BUFFER
      GO TO 115
   80 READ(8,1000) BUFFER
      GO TO 115
   90 READ(9,1000) BUFFER
      GO TO 115
  100 READ(10,1000) BUFFER
  115 DO 210 I=1,80
          J = BUFFER(I)
          J = ICON(J)
          GO TO (215,220,225),L
  215     IBUFF(I) = ITRAN(J)
          GO TO 210
  220     SOBUFF(I) = ITRAN(J)
          GO TO 210
  225     PABUFF(I) = ITRAN(J)
  210 CONTINUE
 1000 FORMAT(80A1)
      RETURN
      END
```

```
      SUBROUTINE WRITEL(NSPACE)
C
CCCCC ***************************************************
CCCCC * WRITEL OUTPUTS THE CURRENT OUTPUT BUFFER TO THE    *
CCCCC * INDICATED FILE.  THE PARAMETER NSPACE WILL INDICATE*
CCCCC * THE NUMBER OF BLANK LINES TO FOLLOW THE OUTPUT     *
CCCCC * BUFFER.                                           *
CCCCC ***************************************************
C
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      IF(OBP .EQ. 0) GO TO 999
      NBLANK = 1
      DO 5 I=1,OBP
          J = OBUFF(I)
          IF(J .NE. 1) NBLANK = I
          OBUFF(I) = OTRAN(J)
    5 CONTINUE
      OBP = IMIN(120,NBLANK)
      GO TO (10,30,40,50,60,70,80), OFILE
   10 WRITE(6,1002) (OBUFF(I),I=1,OBP)
   11 IF(NSPACE .EQ. 0) GO TO 20
      DO 15 I=1,NSPACE
          WRITE(6,1001)
   15 CONTINUE
      GO TO 20
   30 WRITE(4,1000) (OBUFF(I),I=1,OBP)
      GO TO 11
   40 WRITE(7,1000) (OBUFF(I),I=1,OBP)
      GO TO 11
   50 WRITE(8,1000) (OBUFF(I),I=1,OBP)
      GO TO 11
   60 WRITE(9,1000) (OBUFF(I),I=1,OBP)
      GO TO 11
   70 WRITE(10,1000) (OBUFF(I),I=1,OBP)
      GO TO 11
   80 WRITE(11,1000) (OBUFF(I),I=1,OBP)
      GO TO 11
   20 OBP = 0
  999 RETURN
 1000 FORMAT(80A1)
 1001 FORMAT(1H )
 1002 FORMAT(1H ,120A1)
      END
```

```
      SUBROUTINE PAD(LC,CHR,I)
C
CCC   ***************************************************
CCC   * PAD PLACES A SPECIFIED NUMBER OF DESIRED CHARACTERS*
CCC   * INTO THE OUTPUT BUFFER.                          *
CCC   ***************************************************
C
      INTEGER LC,CHR,I,T(20)
      J = IMIN(I,20)
      DO 10 K=1,J
         T(K) = CHR
   10 CONTINUE
      CALL FORM(LC,T,1,J,20,.FALSE.)
      RETURN
      END
```

```
      SUBROUTINE CONOUT(LC,K,N,BASE)
C     ********************************************************
C     * CONOUT PLACES ANY INTEGER CONSTANT IN ANY OF THE     *
C     * FOUR BASES AVAILABLE AND WITH A SPECIFIED FIELD      *
C     * WIDTH LESS THAN EQUAL TO 19.  THIS FIELD IS FILLED   *
C     * WITH ZEROES OR BLANKS, DEPENDING ON THE SIGN OF K    *
C     ********************************************************
C
      INTEGER K,N,BASE,T(20),LC
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER TEXT (138)
      COMMON /TEXTS/ TEXT
      LOGICAL AFLAG,TOGSET
      COMMON /FLAGS/ AFLAG,TOGSET
      LOGICAL ZSUP
      NP = N
      ZSUP = K .LT. 0
      KP = IMIN(IABS(K),19)
      DO 10 I=1,KP
          T(I) = 1
   10 CONTINUE
      IP = KP + 1
      DO 20 I=1,KP
          T(IP-I) = MOD(NP,BASE) + 2
          NP = NP/BASE
          IF(ZSUP .AND. (NP .EQ. 0)) GO TO 30
   20 CONTINUE
   30 IF(.NOT. AFLAG) GO TO 35
      J = OBP + I + 1
      IF(J .LE. 71) GO TO 35
      CALL WRITEL(0)
C     OUTPUT('      *'/-/)
      CALL FORM(0,TEXT ,10,11,6,.TRUE.)
   35 IF(BASE .EQ. 8) GO TO 40
      IF(BASE .EQ. 2) GO TO 45
      IF(BASE .NE. 16) GO TO 50
      KP = KP + 1
      T(KP) = 19
      GO TO 50
   40 KP = KP + 1
      T(KP) = 28
      GO TO 50
   45 KP = KP + 1
      T(KP) = 13
   50 CALL FORM(LC,T,1,KP,20,.FALSE.)
      RETURN
      END
```

```
      SUBROUTINE FORM(LC,CHARS,START,FINISH,LENGTH,UNPAC)
C
CCC   ***************************************************
CCC   * FORM UNPACKS DATA IF NECESSARY AND PLACES IT INTO  *
CCC   * THE OUTPUT BUFFER.                                 *
CCC   ***************************************************
C
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER MAXPAC,WDSIZE,LINENB
      COMMON /INIT/ MAXPAC,WDSIZE,LINENB
      INTEGER CHARS(LENGTH),LC,START,FINISH,DUMLEN
      LOGICAL UNPAC
      DUMLEN = LENGTH
      J = START
      I = LC + 1
      GO TO (100,200,300),I
  100 CALL WRITEL(0)
  200 IF(UNPAC) GO TO 250
      IF(J .GT. FINISH) GO TO 999
      OBP = OBP + 1
      OBUFF(OBP) = CHARS(J)
      J = J + 1
      IF (OBP .GE. MARGIN) GO TO 100
      GO TO 200
  250 N = CONTRL(19)/6
  260 IF(N .LE. DUMLEN) GO TO 270
      N = MOD(DUMLEN,N)
  270 K = OBP + N
      IF (K .GT. MARGIN) CALL WRITEL(0)
      OBP = OBP + N
      CALL UNPACK(CHARS(J),N)
      DUMLEN = DUMLEN - N
      J = J + 1
      IF(J .GT. FINISH) GO TO 999
      GO TO 260
  300 IF(J .GT. FINISH) GO TO 999
      IF(CHARS(J) .NE. 1) GO TO 200
      J = J + 1
      GO TO 300
  999 RETURN
      END
```

```
      SUBROUTINE PRSYM(CC,SYM)
C
CCC   ***************************************************
CCC   * PRSYM PRINTS THE APPROPRIATE SYMBOLS FOR ERROR  *
CCC   * MESSAGES.                                       *
CCC   ***************************************************
C
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      INTEGER CC,SYM,SHL,SHR,RIGHT
      INTEGER PBUFF(29)
      K = VLOC(SYM+1)
      IF(SYM .GT. NT) GO TO 100
      L = V(K)
      CALL FORM(CC,V,K+1,K+L,NSY+1,.FALSE.)
      GO TO 999
  100 L = RIGHT(K,15) - 1
      K = SHR(K,15)
      KP = 0
      DO 200 I=1,K,PACK
           L = L + 1
           LP = V(L)
           JP = PACK * 6
           DO 200 J=1,PACK
                JP = JP - 6
                KP = KP + 1
                IP = SHR(LP,JP)
                PBUFF(KP) = RIGHT(IP,6) + 1
  200 CONTINUE
      CALL FORM(CC,PBUFF,1,K,30)
  999 RETURN
      END
```

```
      SUBROUTINE UNPACK(WORD,I)
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER MAXPAC,WDSIZE,LINENB
      COMMON /INIT/ MAXPAC,WDSIZE,LINENB
      INTEGER SHL,SHR,SHLCNT,WORD,I
      SHLCNT = 0
      DO 100 J=1,I
          K = SHL(WORD,CONTRL(19)-6*J)
          IF(K .GE. 0) GO TO 80
   50     IF((K .GE. 0).OR.(SHLCNT.GE.6)) GO TO 60
          SHLCNT = SHLCNT + 1
          K = SHL(WORD,CONTRL(19)+SHLCNT-6*J)
          GO TO 50
   60     K = SHR(K,CONTRL(19)+SHLCNT-6)
          DO 70 L=1,SHLCNT
              K = K + 2**(6-L)
   70     CONTINUE
          SHLCNT = 0
          GO TO 90
   80     K = SHR(K,CONTRL(19)-6)
   90     OBUFF(OBP-J+1) = K
  100 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE CLOOP
C
CCC   ***************************************************
CCC   * CLOOP CONTROLS THE PROGRAM FLOW WHEN STACKING   *
CCC   * DECISIONS AND REDUCTION DECISIONS ARE MADE.     *
CCC   ***************************************************
C
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,COMPIL
      INTEGER ACCLEN,ACCUM(99),NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR(75),VARTOP,VARC(256),MVAR
      COMMON /SCANN/ACCLEN,ACCUM,NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR,VARTOP,VARC,MVAR
      INTEGER SHL,SHR,RIGHT
      LOGICAL STACK
      COMPIL = .TRUE.
   10 IF(.NOT. COMPIL) GO TO 999
      IF(.NOT.STACK(0)) GO TO 30
C
CCC   STACK MAY HAVE SET COMPIL TO FALSE
C
      IF(.NOT. COMPIL) GO TO 999
      SP = SP + 1
      IF(SP .LT. MSTACK) GO TO 20
      CALL ERROR(2)
      GO TO 999
   20 PSTACK(SP) = TOKEN
C
CCC   INSERT ACCUM INTO VARC
C
      IF((TOKEN.NE.STRV).AND.(TOKEN.NE.NUMBV).AND.(TOKEN.NE.IDENTV))
     1GO TO 45
      VAR(SP) = VARTOP
      IF(ACCLEN .EQ. 0) GO TO 999
      DO 40 I=1,ACCLEN
          VARC(VARTOP) = ACCUM(I)
          VARTOP = VARTOP + 1
          IF(VARTOP .LE.MVAR) GO TO 40
```

```
            CALL ERROR(3)
            VARTOP = 1
 40         CONTINUE
 45         CONTINUE
      CALL SCAN
      GO TO 10
 30   CALL REDUCE
      GO TO 10
999   RETURN
      END
```

```
      LOGICAL FUNCTION STACK(Q)
      INTEGER GETC1,SHL,SHR
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,COMPIL
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,PROCV,SEMIV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PPTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,PROCV,SEMIV,EOFILE
      I = GETC1(PSTACK(SP),TOKEN)+1
   50 GO TO (100,200,300,400),I
C     ILLEGAL SYMBOL PAIR
  100 CALL ERROR(1)
      CALL PRSYM(0,PSTACK(SP))
      CALL PAD(1,1,1)
      CALL PRSYM(1,TOKEN)
      CALL WRITEL(2)
      COMPIL = .FALSE.
      RETURN
C
CC    RETURN TRUE
C
  200 STACK = .TRUE.
      GO TO 999
C
CC    RETURN FALSE
C
  300 STACK = .FALSE.
      GO TO 999
  400 J = SHL(PSTACK(SP-1),16)+SHL(PSTACK(SP),8)+TOKEN
      IU = NC1TRI+2
      IL = 1
  450 K = SHR(IU+IL,1)
      JP = C1TRI(K)
      IF(J .LT. JP) IU=K
      IF(J .GE. JP) IL=K
      IF((IU-IL) .GT. 1) GO TO 450
C
CC    CHECK FOR MATCH
C
      STACK = J .EQ. C1TRI(IL)
  999 RETURN
      END
```

```
      INTEGER FUNCTION GETC1(I,J)
      INTEGER SHL,SHR,RIGHT
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,COMPIL
      K = (NT+1)*I+J
      L = K/15+1
      L = C1(L)
      M = SHL(14-MOD(K,15),1)
      GETC1 = RIGHT(SHR(L,M),2)
      RETURN
      END
```

```
      LOGICAL FUNCTION PROK(PRD)
      INTEGER SHL,SHR,RIGHT,GETC1
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,COMPIL
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      INTEGER PRD
C
C     CHECK FOR EQUAL OR EMBEDDED RIGHT PARTS
C
      I = CONTC(PRD) + 1
      GO TO (100,200,300,400),I
C
C     NO CHECK REQUIRED
C
  100 PROK = .TRUE.
      RETURN
C
C     RIGHT CONTEXT CHECK
C
  200 PROK = GETC1(HDTB(PRD),TOKEN) .NE. 0
      RETURN
C
C     LEFT CONTEXT CHECK
C
  300 K = HDTB(PRD) - NT
      L = PRLEN(PRD)
      I = PSTACK(SP-L)
      L = LEFTI(K) + 1
      LP = LEFTI(K+1)
      IF(L .GT. LP) GO TO 350
      DO 350 J=L,LP
          IF(LEFTC(J) .NE. I) GO TO 350
          PROK = .TRUE.
          RETURN
  350 CONTINUE
      PROK = .FALSE.
      RETURN
C
C     CHECK TRIPLES
```

```
C
  400 K = HDTB(PRD) -- NT
      L = PRLEN(PRD)
      I = SHL(PSTACK(SP-L),8) + TOKEN
      L = TRIPI(K) + 1
      LP = TRIPI(K+1)
      IF(L .LT. LP) GO TO 450
      DO 450 J=L,LP
          IF(CONTT(J) .NE. I) GO TO 450
          PROK = .TRUE.
          RETURN
  450 CONTINUE
      PROK = .FALSE.
      RETURN
      END
```

```
      SUBROUTINE REDUCE
      INTEGER PRD,SHL,SHR,RIGHT,J,M
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,COMPIL
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,PROCV,SEMIV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,PROCV,SEMIV,EOFILE
      LOGICAL JL,ML,PROK
      EQUIVALENCE (J,JL),(M,ML)
C
C     PACK THE TOP OF THE STACK
C
      K = SP - 4
      L = SP - 1
      J = 0
      DO 10 I=K,L
          J = SHL(J,8) + PSTACK(I)
   10 CONTINUE
      K = PRIND(PSTACK(SP)) + 1
      L = PRIND(PSTACK(SP)+1)
      DO 20 PRD=K,L
          M = PRLEN(PRD)
          M = PRMASK(M)
          ML = (ML .AND. JL)
          IF(M .NE. PRTB(PRD)) GO TO 20
          IF(.NOT. PROK(PRD)) GO TO 20
          MP = SP - PRLEN(PRD) + 1
          MPP1 = MP + 1
          J = HDTB(PRD)
          CALL SYNTH(PRDTB(PRD),J)
          SP = MP
          PSTACK(SP) = J
          RETURN
   20 CONTINUE
C
C     NO APPLICABLE PRODUCTION
      FAILSF = .FALSE.
      CALL ERROR(4)
      COMPIL = .FALSE.
      RETURN
      END
```

```
      SUBROUTINE SCAN
      INTEGER GNC
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,COMPIL
      INTEGER ACCLEN,ACCUM(99),NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR(75),VARTOP,VARC(256),MVAR
      COMMON /SCANN/ACCLEN,ACCUM,NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR,VARTOP,VARC,MVAR
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
C
CC    SCAN GETS THE NEXT ENTITY FROM THE INPUT STREAM. IT ALSO TAGS
CC    THEM WITH A TYPE AND TOKEN VALUE.
CC
CC         TYPE                ITEM        VARIABLE
CC          1                END OF FILE    EOFLAG
CC          2                IDENTIFIER     IDENT
CC          3                  NUMBER       NUMB
CC          4                  STRING       STR
CC          5                  CONCAT       // OR /-/
C           6                 SPEC CHAR     SPECL
      ACCLEN = 0
   10 I = GNC(0)
      IF(I .EQ. 0) GO TO 50
      GO TO (10,200,200,200,200,200,200,200,200,200,200,300,
     1       300,300,300,300,300,300,300,300,300,300,300,300,
     2       300,300,300,300,300,300,300,300,300,300,300,300,
     3       300,400,400,400,400,400,400,400,400,200,400,400,400,
     4       400,400,400,400,400,400,400,400,400,400,400,400,
     5       400,400,400,400),I
C
CC    END OF FILE
C
   50 TYPE = EOFLAG
      TOKEN = EOFILE
      GO TO 5600
C     NUMBER FOUND
  200 TYPE = NUMB
```

```
  210 ACCLEN = ACCLEN + 1
      ACCUM(ACCLEN) = I
      I = GNC(0)
      IF((I.GE. 2).AND.(I .LE. 11)) GO TO 210
      IBP = IBP - 1
      TOKEN = NUMBV
      GO TO 5600
C
CC    IDENTIFIER
C
  300 TYPE = IDENT
  310 ACCLEN = ACCLEN + 1
      ACCUM(ACCLEN) = I
      I = GNC(0)
      IF((I .GE. 2).AND.(I .LE. 38)) GO TO 310
      IBP = IBP - 1
      IF(ACCLEN .LE. 6) GO TO 315
      CALL ERROR(7)
      CALL FORM(0,ACCUM,1,ACCLEN,ACCLEN,.FALSE.)
      CALL WRITEL(1)
      COMPIL = .FALSE.
      RETURN
  315 STYPE = 0
C
CC    GO LOOK FOR SPECIAL TOKENS
C
      GO TO 5000
C
CC    SPECIAL CHARACTER FOUND
C
  400 IF(I .EQ. 46) GO TO 500
      IF(I .EQ. 41) GO TO 450
      TYPE = SPECL
      ACCLEN = 1
      ACCUM(1)= I
      GO TO 5000
C
CC    BEGINNING OF CONCATINATION FOUND
C
  450 TYPE = CONCAT
      I = GNC(0)
      IF(I .EQ. 41) GO TO 470
      IF(I .NE. 45) GO TO 460
      TOKEN = NCATV
      I = GNC(0)
  460 IF(I .EQ. 41) GO TO 5600
      CALL ERROR(5)
      COMPIL = .FALSE.
```

```
      RETURN
  470 TOKEN = CATV
      GO TO 5600
C
C     STRING QUOTE FOUND
  500 TYPE = STR
      ACCUM(1) = 1
  510 I = GNC(0)
      IF(I .EQ. 46) GO TO 530
  520 ACCLEN = ACCLEN + 1
      IF(ACCLEN .LE. 99) GO TO 525
      CALL ERROR(6)
      RETURN
  525 ACCUM(ACCLEN) = I
      GO TO 510
C
C     STRING QUOTE FOUND, CHECK FOR END
C
  530 I = GNC(0)
      IF(I .EQ. 46) GO TO 520
      IBP = IBP - 1
      STYPE = 0
      TOKEN = STRV
      GO TO 5600
C
C     CHECK FOR SPECIAL TOKENS
C
 5000 TOKEN = 0
      J = VINDX(ACCLEN) + 1
      K = VINDX(ACCLEN+1)
      DO 5100 I=J,K
          L = VLOC(I)
          LP = L + V(L)
          L = L + 1
          N = 1
          DO 5200 M=L,LP
              IF(ACCUM(N) .NE. V(M)) GO TO 5100
 5200         N = N + 1
          CONTINUE
          TOKEN = I - 1
          GO TO 5600
 5100 CONTINUE
      GO TO 5500
 5300 GO TO 5600
 5500 IF(TYPE .NE. IDENT) GO TO 5600
      TOKEN = IDENTV
 5600 RETURN
      END
```

```
      SUBROUTINE SYNTH(PROD,SYM)
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,COMPIL
      INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
     1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
     2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
     3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
     4TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
     1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
     2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
     3PACK,TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
      INTEGER ACCLEN,ACCUM(99),NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR(75),VARTOP,VARC(256),MVAR
      COMMON /SCANN/ACCLEN,ACCUM,NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR,VARTOP,VARC,MVAR
      INTEGER MAXPAC,WDSIZE,LINENB
      COMMON /INIT/ MAXPAC,WDSIZE,LINENB
      INTEGER VERLEN,START,FINISH,LENGTH,PAKCNT,VERSE(500),CC
      COMMON /FORMS/ VERLEN,START,FINISH,LENGTH,PAKCNT,VERSE,CC
      INTEGER RADIX(8),VARB(8),FW(8)
      COMMON /CNOUT/ RADIX,VARB,FW
      INTEGER EQUIV(10),EQPTR,NUMCTR,NUMLN,MAXLN,OBPCK
      COMMON /EQUIVA/ EQUIV,EQPTR,NUMCTR,NUMLN,MAXLN,OBPCK
      INTEGER TEXT (138)
      COMMON /TEXTS/ TEXT
      LOGICAL AFLAG,TOGSET
      COMMON /FLAGS/ AFLAG,TOGSET
      INTEGER VCOPY(37),COMLEN(37),COMVEC(500),CPTR,SOURCE(6),PCNT,
     1        COMFIN(37)
      LOGICAL ERFLAG,SORCE
      COMMON /COPYS/ VCOPY,COMLEN,COMVEC,CPTR,SOURCE,PCNT,ERFLAG,SORCE,
     1               COMFIN
      INTEGER PROD,SYM,SHL,SHR,RIGHT,CONTIN,BASE
      GO TO (100,200,300,400,500,600,700,800,900,1000,1100,1200,1300,
     1       1400,1500,1600,1700,1800,1900,2000,2100,2200,2300,2400,
     2       2500,2600,2700,2800,2900,3000,999,999,999,999,999,999,999,
     3       999,999,999,999,999,999,999,999,999,999,999,4900,5000,5100,
     4       5200),PROD
C       <PROGRAM> ::= <STATEMENT LIST>
100   COMPIL = .FALSE.
      IF(AFLAG) GO TO 999
C     OUTPUT('      CALL WRITEL(0)')
      CALL FORM(0,TEXT ,12,15,20,.TRUE.)
      CALL WRITEL(0)
```

```
      CC = 0
      GO TO 999
C       <STATEMENT LIST> ::= <STATEMENT>
200   COMPIL = .FALSE.
      IF( AFLAG ) GO TO 999
C     OUTPUT('      CALL WRITEL(0)')
      CALL FORM(0,TEXT ,12,15,20,.TRUE.)
      CALL WRITEL(0)
      CC = 0
      GO TO 999
C       <STATEMENT LIST> ::= <STATEMENT LIST> <STATEMENT>
300   CONTINUE
      GO TO 999
C       <STATEMENT> ::= <OUTPUT_STATEMENT>
400   CONTINUE
      GO TO 999
C       <STATEMENT> ::= <APPEND_STATEMENT>
500   CONTINUE
      GO TO 999
C       <STATEMENT> ::= <COPY_STATEMENT>
600   CONTINUE
      GO TO 999
C       <STATEMENT> ::= <CHANGE_STATEMENT>
700   CONTINUE
      GO TO 999
C       <STATEMENT> ::= <INSERT_STATEMENT>
800   CONTINUE
      GO TO 999
C       <STATEMENT> ::= <DELETE_STATEMENT>
900   CONTINUE
      GO TO 999
C       <COPY_STATEMENT> ::= <COPY> <NUMBER>
1000  J = VAR(SP)
      K = VARTOP - 1
      II = 0
      VARTOP = VAR(SP)
      N = K - J
      DO 1010 I=J,K
         II = (VARC(I)-2)*10**N+II
         N = N - 1
1010  CONTINUE
      START = VCOPY(II)
      LENGTH = COMLEN(II)
      FINISH = COMFIN(II)
      CALL FORM(0,COMVEC,START,FINISH,LENGTH,.TRUE.)
      CALL WRITEL(0)
      AFLAG = .TRUE.
      GO TO 999
```

```
C       <OUTPUT_STATEMENT> ::= <OUTPUT_HEAD> )
1100  CONTINUE
      GO TO 999
C       <APPEND_STATEMENT> ::= <APPEND> <SUBJECT>
1200  AFLAG = .TRUE.
      EQPTR = EQPTR + 1
      EQUIV(EQPTR) = NUMCTR
      NTOT=0
      DO 1206 I=1,EQPTR
      NTOT = NTOT + EQUIV(I)
1206  CONTINUE
      IF (EQPTR .EQ. 1) GO TO 1225
C     OUTPUT('      INTEGER '//)
      CALL FORM(0,TEXT ,26,28,14,.TRUE.)
      CONTIN = 0
      DO 1210 I=1,EQPTR
          CONTIN = CONTIN + 1
          IF (CONTIN .LT. 5) GO TO 1209
          CONTIN = 0
          CALL WRITEL(0)
C         OUTPUT('     *'//)
      CALL FORM(0,TEXT ,29,30,6,.TRUE.)
1209      II = I - 1
          III = EQUIV(I)
          IF (I .NE. EQPTR) GO TO 1208
          III = 500 - NTOT + EQUIV(I)
C         OUTPUT(//'VERSE'/-/(-4) II:10//'('/-/(-10) III:10//')'//)
1208  CALL FORM(1,TEXT ,31,31,5,.TRUE.)
          CALL CONOUT(2,-4    ,II    ,10    )
      CALL FORM(1,TEXT ,32,32,1,.TRUE.)
          CALL CONOUT(2,-10   ,III   ,10    )
      CALL FORM(1,TEXT ,33,33,1,.TRUE.)
          IF (I .EQ. EQPTR) GO TO 1210
C         OUTPUT(//',')
      CALL FORM(1,TEXT,34,34,1,.TRUE.)
1210  CONTINUE
      CALL WRITEL(0)
C     OUTPUT('      EQUIVALENCE (VERSE(1),VERSEO(1)),'//)
      CALL FORM(0,TEXT ,35,42,39,.TRUE.)
      J = 1
      CONTIN = 2
      K = EQPTR - 1
      DO 1220 I=1,K
          CONTIN = CONTIN + 1
          IF (CONTIN .LT. 4) GO TO 1222
          CONTIN = 0
          CALL WRITEL(0)
C         OUTPUT('     *'//)
```

```
      CALL FORM(0,TEXT ,43,44,6,.TRUE.)
1222      J = J + EQUIV(I)
C        OUTPUT(//'(VERSE('/-/(-10) J:10//'),VERSE'/-/(-4) I:10//)
      CALL FORM(1,TEXT ,45,46,7,.TRUE.)
      CALL CONOUT(2,-10    ,J      ,10     )
      CALL FORM(1,TEXT ,47,48,7,.TRUE.)
          CALL CONOUT(2,-4     ,I      ,10     )
C         OUTPUT(//'(1))'//)
      CALL FORM(1,TEXT ,49,49,4,.TRUE.)
              IF (I .EQ. K) GO TO 1220
C         OUTPUT(//',')
      CALL FORM(1,TEXT,34,34,1,.TRUE.)
1220  CONTINUE
      CALL WRITEL(0)
1225  KK = 0
      DO 1230 J=1,EQPTR
          IF (EQPTR .EQ. 1) GO TO 1235
          JJ = J - 1
C         OUTPUT('      DATA VERSE'/-/(-4) JJ:10//'/')
      CALL FORM(0,TEXT ,50,53,16,.TRUE.)
          CALL CONOUT(2,-4     ,JJ     ,10     )
      CALL FORM(1,TEXT ,54,54,1,.TRUE.)
          CALL WRITEL(0)
          GO TO 1236
C         OUTPUT('      DATA VERSE/')
1235  CALL FORM(0,TEXT,55,58,17,.TRUE.)
          CALL WRITEL(0)
C         OUTPUT('     *'/-/)
1236  CALL FORM(0,TEXT ,59,60,6,.TRUE.)
          IF (J .EQ. 1) GO TO 1237
          KK = KK + EQUIV(J-1)
1237      K = EQUIV(J)
          DO 1240 I=1,K
              IVAL = VERSE(I+KK)
C             OUTPUT(/-/(-20) IVAL:10//)
              CALL CONOUT(2,-20    ,IVAL   ,10     )
              IF (I .EQ. K) GO TO 1240
C             OUTPUT(//',')
      CALL FORM(1,TEXT,34,34,1,.TRUE.)
1240      CONTINUE
          IF (J .EQ. EQPTR) GO TO 1230
C         OUTPUT(//'/'//)
      CALL FORM(1,TEXT ,61,61,1,.TRUE.)
1230  CONTINUE
      J = 500 - NTOT
C         OUTPUT(//',')
      CALL FORM(1,TEXT,34,34,1,.TRUE.)
C     OUTPUT('     *'//(-4)J:10//'*0/')
```

```
      CALL FORM(0,VERSE,16,17,6,.TRUE.)
      CALL CONOUT(1,-4    ,J     ,10    )
      CALL FORM(1,VERSE,18,18,3,.TRUE.)
      CALL WRITEL(0)
      GO TO 999
C       <OUTPUT_HEAD> ::= <OUTPUT> (
1300  CONTINUE
      GO TO 999
C       <OUTPUT_HEAD> ::= <OUTPUT_HEAD> <STRING>
1400  VERLEN = VERLEN + 1
      AFLAG = .FALSE.
      START = VERLEN
      LENGTH = 0
      PAKCNT = 1
      J = VAR(SP)
      K = VARTOP - 1
      VARTOP = VAR(SP)
      DO 20 I=J,K
          IF(PAKCNT .LE. MAXPAC) GO TO 10
          PAKCNT = 1
          VERLEN = VERLEN + 1
   10     KV = SHL(VERSE(VERLEN),6)
          VERSE(VERLEN) = KV + VARC(I)
          PAKCNT = PAKCNT + 1
          LENGTH = LENGTH + 1
   20 CONTINUE
      FINISH = VERLEN
C     OUTPUT('      CALL FORM('//(1)CC:10//',VERSE,'/-/(-4)START:10//)
C     OUTPUT(//',//-/(-4)FINISH:10//',//-/(-4)LENGTH:10//',.TRUE.)')
      CALL FORM(0,TEXT ,62,65,16,.TRUE.)
      CALL CONOUT(1,1     ,CC    ,10    )
      CALL FORM(1,TEXT ,66,67,7,.TRUE.)
      CALL CONOUT(2,-4    ,START ,10    )
      CALL FORM(1,TEXT ,68,68,1,.TRUE.)
      CALL CONOUT(2,-4    ,FINISH,10    )
      CALL FORM(1,TEXT ,69,69,1,.TRUE.)
      CALL CONOUT(2,-4    ,LENGTH,10    )
      CALL FORM(1,TEXT ,70,71,8,.TRUE.)
      CALL WRITEL(0)
      BASE = 10
      DO 30 I=START,FINISH
          J = VERSE(I)
          NDIGIT = 1
   31     J = J / BASE
          IF (J .EQ. 0) GO TO 33
          NDIGIT = NDIGIT + 1
          GO TO 31
   33     JJ = OBPCK + NDIGIT + 1
```

```
            IF (JJ .LE. 71) GO TO 37
            JJJ = NUMLN + 1
            IF (JJJ .LE. MAXLN) GO TO 35
            NUMLN = 0
            EQPTR = EQPTR + 1
            EQUIV(EQPTR) = NUMCTR
            NUMCTR = 0
   35       OBPCK = 6
            NUMLN = NUMLN + 1
   37       OBPCK = OBPCK + NDIGIT + 1
            NUMCTR = NUMCTR + 1
   30 CONTINUE
      GO TO 999
C       <OUTPUT_HEAD> ::= <OUTPUT_HEAD> <CONCAT>
1500  CONTINUE
      CC=1
      AFLAG = .TRUE.
      GO TO 999
C       <OUTPUT_HEAD> ::= <OUTPUT_HEAD> <NCONCAT>
1600  CONTINUE
      CC=2
      AFLAG = .TRUE.
      GO TO 999
C       <OUTPUT_HEAD> ::= <OUTPUT_HEAD> <CONOUT_CALL>
1700  CONTINUE
C     OUTPUT('      CALL CONOUT('/-/(-3)CC:10/-/)
      CALL FORM(0,TEXT ,72,75,18,.TRUE.)
      CALL CONOUT(2,-3    ,CC    ,10    )
      CALL PAD(1,48,1)
      CALL FORM(1,FW,1,6,8,.FALSE.)
      CALL PAD(1,48,1)
      CALL FORM(1,VARB,1,6,8,.FALSE.)
      CALL PAD(1,48,1)
      CALL FORM(1,RADIX,1,6,8,.FALSE.)
      CALL PAD(1,43,1)
      CALL WRITEL(0)
      AFLAG = .FALSE.
      GO TO 999
C       <CONOUT_CALL> ::= <CONOUT_STATEMENT>
1800  CONTINUE
      GO TO 999
C       <CONOUT_CALL> ::= <CONOUT_CALL> <RADIX>
1900  CONTINUE
      GO TO 999
C       <CONOUT_STATEMENT> ::= <IDENTIFIER>
2000  GO TO 2200
C       <CONOUT_STATEMENT> ::= <NUMBER>
2100  GO TO 2200
```

```
C       <CONOUT_STATEMENT> ::= <FIELD_WIDTH> <IDENTIFIER>
2200  J = VAR(SP)
      K = 1
      L = VARTOP - 1
      DO 13901 I =J,L
          VARB(K) = VARC(I)
          K = K + 1
13901 CONTINUE
      IJ = VARTOP - J + 1
      DO 13902 I=IJ,8
          VARB(I) = 1
13902 CONTINUE
      VARTOP = VAR(SP)
      GO TO 999
C       <CONOUT_STATEMENT> ::= <FIELD_WIDTH> <NUMBER>
2300  GO TO 2200
C       <FIELD_WIDTH> ::= <FW_HEAD> )
2400  CONTINUE
      GO TO 999
C       <FW_HEAD> ::= (
2500  CONTINUE
      GO TO 999
C       <FW_HEAD> ::= <FW_HEAD> <IDENTIFIER>
2600  J = VAR(SP)
      K = 1
      L = VARTOP - 1
      DO 14301 I=J,L
          FW(K) = VARC(I)
          K = K + 1
14301 CONTINUE
      IJ = VARTOP - J + 1
      DO 14302 I=IJ,8
          FW(I) = 1
14302 CONTINUE
      VARTOP = VAR(SP)
      GO TO 999
C       <FW_HEAD> ::= <FW_HEAD> <NUMBER>
2700  GO TO 2600
C       <RADIX> ::= <RADIX_HEAD> <NUMBER>
2800  J = VAR(SP)
      K = 1
      L = VARTOP - 1
      DO 14501 I=J,L
          RADIX(K) = VARC(I)
          K = K + 1
14501 CONTINUE
      IJ = VARTOP - J + 1
      DO 14502 I=IJ,8
```

```
            RADIX(I) = 1
14502 CONTINUE
      VARTOP = VAR(SP)
      GO TO 999
C       <RADIX> ::= <RADIX_HEAD> <IDENTIFIER>
2900  GO TO 2800
C       <RADIX_HEAD> ::= :
3000  CONTINUE
      GO TO 999
C       <OUTPUT> ::= OUTPUT
4900  CONTINUE
      GO TO 999
C       <APPEND> ::= APPEND
5000  CONTINUE
      GO TO 999
C       <COPY> ::= COPY
5100  CONTINUE
      GO TO 999
C       <SUBJECT> ::= VERSE
5200  CONTINUE
  999 RETURN
      END
```

```
      SUBROUTINE MERG
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER PABUFF(80),SOBUFF(80),SONUM(8),PANUM(8),PASNUM(8),MERGE
      LOGICAL INSERT,FAULT
      COMMON /MERGES/ PABUFF,SOBUFF,SONUM,PANUM,PASNUM,MERGE,INSERT,
     1FAULT
      LOGICAL NINES,MERGIT
   50 IF(PABUFF(9) .NE. 47) GO TO 200
      CALL SEQNUM(2)
  100 IFILE = CONTRL(20)
  150 CALL RDTEXT(2)
      CALL SEQNUM(3)
      IF(MERGIT(SONUM,PASNUM)) GO TO 150
      IFILE = CONTRL(1)
      CALL RDTEXT(3)
      CALL SEQNUM(1)
      IF(NINES(1)) CONTRL(24) = 0
      IF(MERGIT(PANUM,SONUM)) GO TO 50
      DO 175 I = 1,80
         IBUFF(I) = SOBUFF(I)
  175 CONTINUE
      RETURN
  200 INSERT = .TRUE.
      DO 210 I=1,72
         IBUFF(I) = PABUFF(I+8)
  210 CONTINUE
      DO 220 I=73,80
         IBUFF(I) = PABUFF(I-72)
  220 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE SEQDK
      INTEGER IBUFF(30),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER SEQN
      LOGICAL SEQ
      COMMON /SEQNS/ SEQN,SEQ
      IF (SEQ) GO TO 10
      SEQ = .TRUE.
      SEQN = CONTRL(13)
      GO TO 11
   10 SEQN = SEQN + CONTRL(15)
   11 ISEQN = SEQN
      DO 131 I=1,8
          IBUFF(81-I) = MOD(ISEQN,10) + 2
          ISEQN = ISEQN/10
  131 CONTINUE
      RETURN
      END
```

```
      SUBROUTINE COPY
      INTEGER VCOPY(37),COMLEN(37),COMVEC(500),CPTR,SOURCE(6),PCNT,
     1        COMFIN(37)
      LOGICAL ERFLAG,SORCE
      COMMON /COPYS/ VCOPY,COMLEN,COMVEC,CPTR,SOURCE,PCNT,ERFLAG,SORCE,
     1               COMFIN
      INTEGER MAXPAC,WDSIZE,LINENB
      COMMON /INIT/ MAXPAC,WDSIZE,LINENB
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER PABUFF(80),SOBUFF(80),SONUM(8),PANUM(8),PASNUM(8),MERGE
      LOGICAL INSERT,FAULT
      N = CONTRL(19)/6
    5 J = IBUFF(3) - 2
      IF(IBUFF(4) .EQ. 1) GO TO 30
      J = J*10+(IBUFF(4)-2)
   30 DO 10 I=1,5
         IBUFF(I) = 1
   10 CONTINUE
      CPTR = CPTR + 1
      VCOPY(J) = CPTR
      L = 0
      PCNT = 1
   40 DO 50 I=1,80
         IF(PCNT .LE. N) GO TO 45
         PCNT = 1
         CPTR = CPTR + 1
   45    K = SHL(COMVEC(CPTR),6)
         COMVEC(CPTR) = K + IBUFF(I)
         PCNT = PCNT + 1
         L = L + 1
   50 CONTINUE
      COMFIN(J) = CPTR
      IBP = 81
      I = GNC(0)
      IF(CONTRL(24) .EQ. 0) GO TO 60
      IF(SOBUFF(1) .NE. CONTRL(31)) GO TO 40
      COMLEN(J) = L
      DO 55 I=1,80
         IBUFF(I) = SOBUFF(I)
   55 CONTINUE
      GO TO 999
   60 IF(IBUFF(1) .NE. CONTRL(31)) GO TO 40
      COMLEN(J) = L
  999 RETURN
      END
```

```
      SUBROUTINE SETTOG
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER CONTRL(64),IFILE,OFILE
      COMMON /CONTRL/ CONTRL,IFILE,OFILE
      INTEGER VCOPY(37),COMLEN(37),COMVEC(500),CPTR,SOURCE(6),PCNT,
     1        COMFIN(37)
      LOGICAL ERFLAG,SORCE
      COMMON /COPYS/ VCOPY,COMLEN,COMVEC,CPTR,SOURCE,PCNT,ERFLAG,SORCE,
     1               COMFIN
      LOGICAL AFLAG,TOGSET
      COMMON /FLAGS/ AFLAG,TOGSET
      INTEGER PATCH(5)
      DATA PATCH/27,12,31,14,19/
      JJ = 3
      IF(IBUFF(JJ) .NE. 1) GO TO 10
    5 IF(IBUFF(JJ) .NE. 1) GO TO 7
      JJ = JJ + 1
      KK = JJ
      GO TO 5
    7 DO 9 I=1,6
        IF(SOURCE(I) .NE. IBUFF(JJ)) GO TO 15
        JJ = JJ + 1
    9 CONTINUE
      SORCE = .TRUE.
      RETURN
   15 DO 16 I=1,5
        IF(PATCH(I) .NE. IBUFF(KK)) GO TO 999
        KK = KK + 1
   16 CONTINUE
      TOGSET = .TRUE.
      RETURN
   10 J = IBUFF(3)
      DO 20 I=4,80
        II = I
        IF(IBUFF(I) .EQ. 39) GO TO 30
   20 CONTINUE
      GO TO 999
   30 K = 0
      II = II + 1
      DO 40 I=II,80
        L = IBUFF(I)
        IF(L .LE. 1) GO TO 40
        IF(L .GT. 11) GO TO 50
        K = K*10+(L-2)
   40 CONTINUE
   50 CONTRL(J) = K
      RETURN
```

```
999 CALL ERROR(8)
    ERFLAG = .TRUE.
    RETURN
    END
```

```
      SUBROUTINE SEQNUM(K)
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      INTEGER PABUFF(80),SOBUFF(80),SONUM(8),PANUM(8),PASNUM(8),MERGE
      LOGICAL INSERT,FAULT
      COMMON /MERGES/ PABUFF,SOBUFF,SONUM,PANUM,PASNUM,MERGE,INSERT,
     1FAULT
      GO TO (5,15,25,35),K
    5 DO 10 I=1,8
          PANUM(I) = PABUFF(I)
   10 CONTINUE
      GO TO 999
   15 DO 20 I=1,8
          PASNUM(I) = PABUFF(9+I)
   20 CONTINUE
      GO TO 999
   25 DO 30 I=1,8
          SONUM(I) = SOBUFF(72+I)
   30 CONTINUE
      GO TO 999
   35 DO 40 I=1,8
          SONUM(I) = IBUFF(72+I)
   40 CONTINUE
  999 RETURN
      END
```



```
      LOGICAL FUNCTION MERGIT(J,K)
      INTEGER J(8),K(8)
      DO 10 I=1,8
          IF(((J(I).GE.12).OR.(K(I).GE.12)).AND.(J(I).NE.K(I))) GO TO 30
          IF(J(I) .NE. K(I)) GO TO 20
   10 CONTINUE
   15 MERGIT = .TRUE.
      GO TO 999
   20 IF(J(I) .LT. K(I)) GO TO 15
   30 MERGIT = .FALSE.
  999 RETURN
      END
```

```
      LOGICAL FUNCTION NINES(K)
      INTEGER PABUFF(80),SOBUFF(80),SONUM(8),PANUM(8),PASNUM(8),MERGE
      LOGICAL INSERT,FAULT
      COMMON /MERGES/ PABUFF,SOBUFF,SONUM,PANUM,PASNUM,MERGE,INSERT,
     1FAULT
      GO TO(10,20),K
   10 DO 15 I=1,8
         IF(PANUM(I) .NE. 11) GO TO 990
   15 CONTINUE
      NINES = .TRUE.
      GO TO 999
   20 DO 25 I=1,8
         IF(SONUM(I) .NE. 11) GO TO 990
   25 CONTINUE
      NINES = .TRUE.
      GO TO 999
  990 NINES = .FALSE.
  999 RETURN
      END
```

```
      FUNCTION IMIN(I,J)
      IF(I .LT. J) GO TO 10
      IMIN = J
      GO TO 999
   10 IMIN = I
  999 RETURN
      END
```

```
      FUNCTION ICON(I)
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      DO 100 K=1,52
         J = K
         IF( I .EQ. OTRAN(K) ) GO TO 200
  100 CONTINUE
      J = 1
  200 ICON = J
      RETURN
      END
```

```
      INTEGER FUNCTION SHL(I,J)
      SHL = I*(2**J)
      RETURN
      END
```

```
      INTEGER FUNCTION SHR(I,J)
      SHR = I/(2**J)
      RETURN
      END
```

```
      INTEGER FUNCTION RIGHT(I,J)
      RIGHT = MOD(I,2**J)
      RETURN
      END
```

```
      SUBROUTINE ERROR(I)
      INTEGER MAXPAC,WDSIZE,LINENB
      COMMON /INIT/ MAXPAC,WDSIZE,LINENB
      INTEGER TEXT(108)
      COMMON /TEXTS/ TEXT
      LINE = LINENB - 1
C     OUTPUT('***** ERROR('//(2)I:10//') NEAR LINE '/-/(-5)LINE  :10)
      CALL FORM(0,TEXT ,80,82,12,.TRUE.)
      CALL CONOUT(1,2     ,I     ,10    )
      CALL FORM(1,TEXT ,83,85,12,.TRUE.)
      CALL CONOUT(2,-5    ,LINE  ,10    )
      CALL WRITEL(0)
      GO TO(100,200,300,400,500,600,700,800),I
  100 CONTINUE
C     OUTPUT('***** ILLEGAL SYMBOL PAIR. SYMBOLS ARE LISTED BELOW')
      CALL FORM(0,TEXT ,86,96,51,.TRUE.)
      CALL WRITEL(0)
      GO TO 999
  200 CONTINUE
C     OUTPUT('***** PARSE STACK OVERFLOW')
      CALL FORM(0,TEXT ,97,102,26,.TRUE.)
      CALL WRITEL(0)
      GO TO 999
  300 CONTINUE
C     OUTPUT('***** TABLE OVERFLOW')
      CALL FORM(0,TEXT ,103,106,20,.TRUE.)
      CALL WRITEL(0)
      GO TO 999
  400 CONTINUE
C     OUTPUT('***** IMPROPERLY FORMED STATEMENT')
      CALL FORM(0,TEXT ,107,113,33,.TRUE.)
      CALL WRITEL(0)
      GO TO 999
  500 CONTINUE
C     OUTPUT('***** INCORRECT FORMAT FOR CONCATENATION USED')
      CALL FORM(0,TEXT ,114,122,45,.TRUE.)
      CALL WRITEL(0)
      GO TO 999
  600 CONTINUE
C     OUTPUT('***** STRING LENGTH EXCEEDS LIMITS')
      CALL FORM(0,TEXT ,123,129,34,.TRUE.)
      CALL WRITEL(0)
      GO TO 999
  700 CONTINUE
C     OUTPUT('***** VARIABLE BELOW EXCEEDS FORTRAN LENGTH LIMITS')
      CALL FORM(0,TEXT,130,139,50,.TRUE.)
      CALL WRITEL(0)
      GO TO 999
```

```
  800 CONTINUE
C     OUTPUT('***** IMPROPERLY FORMED $$CONTROL STATEMENT')
      CALL FORM(0,TEXT ,140,148,43,.TRUE.)
      CALL WRITEL(0)
  999 RETURN
      END
```

```
      BLOCK DATA
      INTEGER PABUFF(80),SOBUFF(80),SONUM(8),PANUM(8),PASNUM(8),MERGE
      LOGICAL INSERT,FAULT
      COMMON /MERGES/ PABUFF,SOBUFF,SONUM,PANUM,PASNUM,MERGE,INSERT,
     1FAULT
      DATA INSERT/.TRUE./,FAULT/.FALSE./
      INTEGER MAXPAC,WDSIZE,LINENB
      COMMON /INIT/ MAXPAC,WDSIZE,LINENB
      DATA LINENB/0/
      INTEGER IBUFF(80),OBUFF(120),IBP,OBP,ITRAN(256),OTRAN(64),MARGIN
      COMMON /FILES/ IBUFF,OBUFF,IBP,OBP,ITRAN,OTRAN,MARGIN
      DATA IBUFF/80*0/,OBUFF/120*0/,IBP/81/,OBP/0/,ITRAN/256*0/
      DATA OTRAN/1H ,1H0,1H1,1H2,1H3,1H4,1H5,1H6,1H7,1H8,1H9,
     1           1HA,1HB,1HC,1HD,1HE,1HF,1HG,1HH,1HI,1HJ,1HK,
     2           1HL,1HM,1HN,1HO,1HP,1HQ,1HR,1HS,1HT,1HU,1HV,
     3           1HW,1HX,1HY,1HZ,
     4           1H$,1H=,1H.,1H/,1H(,1H),1H+,1H-,1H',1H*,1H,,
     5           1H<,1H>,1H:,1H;,12*0/
      LOGICAL AFLAG,TOGSET
      COMMON /FLAGS/ AFLAG,TOGSET
      DATA TOGSET/.FALSE./
      INTEGER SP,PSTACK(75),MSTACK,PRMASK(5),MP,MPP1
      LOGICAL FAILSF,COMPIL
      COMMON/STACKS/SP,PSTACK,MSTACK,PRMASK,MP,MPP1,FAILSF,COMPIL
      DATA PSTACK/75*0/,SP/4/,MSTACK/75/,MP/0/,MPP1/0/
      INTEGER ACCLEN,ACCUM(99),NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR(75),VARTOP,VARC(256),MVAR
      COMMON /SCANN/ACCLEN,ACCUM,NUMB,IDENT,EOFLAG,SPECL,STR,STYPE,TYPE,
     1CONCAT,VAR,VARTOP,VARC,MVAR
      DATA ACCLEN/0/,ACCUM/99*0/,NUMB/3/,IDENT/2/,EOFLAG/1/,STYPE/0/,
     1SPECL/6/,STR/4/,TYPE/0/,CONCAT/5/,VAR/75*0/,VARTOP/1/,VARC/256*0/,
     2MVAR/75/
      INTEGER VERLEN,START,FINISH,LENGTH,PAKCNT,VERSE(500),CC
      COMMON /FORMS/ VERLEN,START,FINISH,LENGTH,PAKCNT,VERSE,CC
      DATA VERLEN/0/,CC/0/
      DATA VERSE/500*0/
      INTEGER EQUIV(10),EQPTR,NUMCTR,NUMLN,MAXLN,OBPCK
      COMMON /EQUIVA/ EQUIV,EQPTR,NUMCTR,NUMLN,MAXLN,OBPCK
      DATA EQUIV/10*0/,EQPTR/0/,NUMCTR/0/,NUMLN/1/,MAXLN/17/,OBPCK/6/
      INTEGER SEQN
      LOGICAL SEQ
      COMMON /SEQNS/ SEQN,SEQ
      DATA SEQ/.FALSE./
      INTEGER TEXT(148)
      COMMON /TEXTS/ TEXT
      INTEGER TEXT0(72),TEXT1(59),TEXT2(17)
      EQUIVALENCE (TEXT(1),TEXT0(1)),(TEXT(73),TEXT1(1)),
     *(TEXT(132),TEXT2(1))
```

```
 DATA TEXT0/
*17043521,23974927,343779184,25495390,342462531,672665803,
*152571969,17043521,65,17043521,47,17043521,20497879,25810207,
*274636971,17043521,47,192681,17043521,65,17043521,20555288,
*442767967,277740097,557963152,17043521,22124496,4785985,17043521,
*47,557963152,42,43,48,17043521,21088276,556889113,239082145,
*276161578,61802576,494469290,965360,17043521,47,713361246,1066,
*734139421,1936,11025131,17043521,20760524,25495390,16,41,
*17043521,20760524,25495390,1065,17043521,47,41,17043521,20497879,
*21342040,42,814024542,1072,48,48,815920992,68139,17043521/
 DATA TEXT1/
*20497879,20555354,133098,17043521,20497879,20555354,133098,
*801045487,21092186,1898,721785868,486896921,1025,801045487,
*2211592,305229918,610326167,23905565,671475992,225015681,
*208994391,343536655,202521122,34,801045487,23906142,268822476,
*240653965,276108762,34,801045487,24953687,268806224,491091618,
*801045487,22120157,443352919,604313245,406909022,523367448,67167,
*801045487,22123418,494207903,21342040,209458266,486860441,
*238154777,209536665,252869743,801045487,24770388,424154576,
*424277185,277668880,259528148,6375390,801045487,25478996/
 DATA TEXT2/
*204829697,222394018,21115792,272490577,443928636,419787801,
*310194263,341919710,801045487,22120157,443352919,604313245,
*406909030,641312351,493449310,523367448,67167/
 INTEGER VCOPY(37),COMLEN(37),COMVEC(500),CPTR,SOURCE(6),PCNT,
1        COMFIN(37)
 LOGICAL ERFLAG,SORCE
 COMMON /COPYS/ VCOPY,COMLEN,COMVEC,CPTR,SOURCE,PCNT,ERFLAG,SORCE,
1               COMFIN
 DATA SOURCE/30,26,32,29,14,16/,VCOPY/37*0/,COMLEN/37*0/,
1     COMVEC/500*0/,SORCE/.FALSE./,ERFLAG/.FALSE./,CPTR/0/
 INTEGER RADIX(8),VARB(8),FW(8)
 COMMON /CNCUT/ RADIX,VARB,FW
 DATA RADIX/8*0/,FW/8*0/,VARB/8*0/
 INTEGER V(215),VLOC(47),VINDX(13),C1(57),C1TRI(1),PRTB(53),
1PRDTB(53),HDTB(53),PRLEN(53),CONTC(53),LEFTC(1),LEFTI(30),
2CONTT(1),TRIPI(30),PRIND(47),NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,PRTBL,
3PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,PACK,
4TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
 COMMON /SYNTAX/V,VLOC,VINDX,C1,C1TRI,PRTB,PRDTB,HDTB,PRLEN,CONTC,
1LEFTC,LEFTI,CONTT,TRIPI,PRIND,NSY,NT,VLEN,VIL,C1W,C1L,NC1TRI,
2PRTBL,PRDTBL,HDTBL,PRLENL,CONCL,LEFTCL,LEFTIL,CONTL,TRIPL,PRIL,
3PACK,TOKEN,IDENTV,NUMBV,STRV,CATV,NCATV,EOFILE
 INTEGER V0(160),V1(55)
 EQUIVALENCE (V(1),V0(1)),(V(161),V1(1))
 DATA V0/18,49,16,29,29,26,29,51,1,31,26,22,16,25,1,39,1,2,50,1,43,
11,42,1,51,2,31,26,3,16,26,17,4,14,26,27,36,5,33,16,29,30,16,6,14,
219,12,25,18,16,6,15,16,23,10,31,16,6,20,25,30,16,29,31,6,26,32,31,
```

```
327,32,31,6,12,27,27,16,25,15,8,49,25,32,24,13,16,29,50,8,49,30,31,
429,20,25,18,50,8,49,14,26,25,14,12,31,50,9,49,25,14,26,25,14,12,
531,50,12,49,20,15,16,25,31,20,17,20,16,29,5C,808818339,822083584,
6812692371,583270400,308298127,406523904,811988890,528158720,
7810387279,477827072,808788696,289345536,809C39247,507449344,
8812238417,472742976,813036308,255192128,809639634,254602304,
9813032158,257750558,822083584,812692371,585966539,247726080,
A811988890,528200847,188420096,808818201,528200523,375066624/
 DATA V1/809579478,250483918,508235776,810387279,477869199,
118842C096,808788696,289387289,244256768,808788696,289387663,
218842C096,809039247,507491097,244256768,809039247,507491471,
318842C096,813032158,257750558,5846878,822083584,808818339,
4997581534,257750558,822083584,811988890,528201566,192476623,
5410718208,808298127,406566750,192476623,410718208,808788696,
628938838Z,192476623,410718208,810387279,477869918,192476623,
7410718208,809039247,507492190,192476623,410718208,808818201,
852820156C,192476623,410718208,810387279,477869907,392782843,
9494450575,389908401/
 DATA VLCC /1,20,22,24,26,29,33,38,44,51,58,65,72,79,88,97,106,116,
1196737,229507,262277,262279,262281,262283,262285,295055,295057,
2295059,360597,393368,426139,426142,426145,426148,426151,426154,
3426157,426160,524467,524471,590011,590015,590019,590023,590027,
4590031,819411/
 DATA VINDX /1,4,5,6,7,8,13,13,16,17,17,17,18/
 DATA C1 /0,305322,705208482,715825152,32896,42641536,70992,0,
113421789C,713031808,2,0,33554432,524288,8,674,581610122,176859818,
21c7772170,707264512,178825216,2793472,16394,536871594,16384,
32c2144,4096,64,1,0,524288,10410,536887296,1052674,581566464,257,
48c886085,355008512,1787c2208,1052928,16384,67108928,67108868,
510410,603996160,1048578,290783232,42641408,666272,10410,536871074,
6713293826,581566464,42641408,176160778,704643072,167772167/
 DATA C1TRI /0/
 DATA PRTB /0,27,33,35,30,9473,22,23,24,21,0,0,34,0,0,0,0,0,0,0,27,
133,35,37,32,29,18,36,0,34,30,46,42,30,30,27,32,29,0,31,20,38,0,30,
20,0,0,0,0,0,0,0,0/
 DATA PRDTB /0,24,32,37,11,43,33,39,45,13,25,30,35,51,52,41,47,48,
149,50,27,34,40,46,23,28,10,44,21,38,14,31,36,15,16,26,22,29,20,19,
212,3,2,17,42,1,6,4,5,7,8,9,18/
 DATA HDTB /0,32,46,34,40,36,33,35,37,30,27,29,42,18,26,23,24,22,
121,20,27,33,35,37,45,19,39,36,45,34,30,43,42,30,30,27,45,19,45,31,
241,38,38,30,44,25,28,28,28,2c,28,28,31/
 DATA PRLEN /0,2,2,2,2,3,2,2,2,2,1,1,2,1,1,1,1,1,1,1,2,2,2,2,2,2,2,
12,1,2,2,2,2,2,2,2,2,2,1,2,2,2,1,2,1,1,1,1,1,1,1,1/
 DATA CONTC /0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
10,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0/
 DATA LEFTC /0/
 DATA LEFTI /0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
10,0,1/
```

```
 DATA CONTT /0/
 DATA TRIPI /0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
10,0,1/
 DATA PRIND /1,5,11,12,13,13,14,15,16,17,18,19,20,29,33,34,35,39,
139,40,40,40.40.40,40,40,41,41,43,43,43,44,44,44,44,44,45,45,46,47,
248,49,50,51,52,53,53/
 DATA NSY /46/, NT /17/, VLEN /214/, VIL /12/, C1W /36/, C1L /56/,
2NC1TRI /0/, PRTBL /52/, PRDTBL /52/, HDTBL /52/, PRLENL /52/,
3CLNCL /52/, LEFTCL /0/, LEFTIL /29/, CONTL /0/, TRIPL /29/,
4PRIL /46/, PACK /5/, TOKEN /0/, IDENTV /17/, NUMBV /13/,
5STRV /14/, CATV /15/, NCATV /16/, EOFILE /5/
 END
```

LIST OF REFERENCES

1. Pool, P. C. and Waite, W. M., "Machine Independent Software," Procedures of the ACM Second Symposium on Operating Systems Principles, p. 19-24, October 1969.

2. Gries, D., Compiler Construction for Digital Computers, p. 64-83, Wiley, 1971.

3. Orgass, R. J. and Waite, W. M., "A Base for a Mobile Programming System," Communications of the ACM, Volume 12, Number 9, p. 507-510, September 1969.

4. McKeeman, W. M., Horning, J. J., and Wortman, D. B., a compiler gnerator, p. 7, Prentice-Hall, 1970.

5. National Technical Information Service, U. S. Department Of Commerce, Programming For Transferability, by J. E. Fleiss, et al, p. 1-44, 72-87, September 1972.

6. Pool, P. C. and Waite, W. M., "Portability and Adaptability," Advanced Course on Software Engineering, p. 183-277, 1973.

7. Halpern, M. I., "Toward a General Processor for Programming Languages," Communications of the ACM, Volume 11, Number 1, p. 15-25, January 1968.

8. Waite, W. M., "The Mobile Programming System: STAGE2," Communications of the ACM, Volume 13, Number 7, p. 415-421, July 1970.

9. Waite, W. M., "A Language Independent Macro Processor," Communications of the ACM, Volume 10, Number 7, p. 433-440, July 1967.

10. Denning, P. J., "Is It Not Time To Define "Structured Programming"?," Operating Systems Review, Volume 8, Number 1, p. 6-7, January 1974.

11. A Guide To PL/M Programming, Revision 1, Intel Corporation, September 1973.

12. Hamlet, R. G., Portability and Adaptability, computing reviews, p. 62-63, February 1974.

BIBLIOGRAPHY

1. Brown, P. J., "The ML/I Macro Processor," Communications of the ACM, Volume 10, Number 10, p. 618-623, October 1967.

2. Cook, A. J., "A User's Guide to MORTRAN," SLAC Computation Group, Stanford, California, CGTM NO. 150, July, 1973.

3. Kildall, G. A., PL/M Compiler Pass-1 Version 2.0, Copyright Intel Corporation, 1973.

4. McIlroy, M. D., "Macro Instruction Extensions of Compiler Languages," Communications of the ACM, volume 3, Number 4, p. 214-220, April 1960.

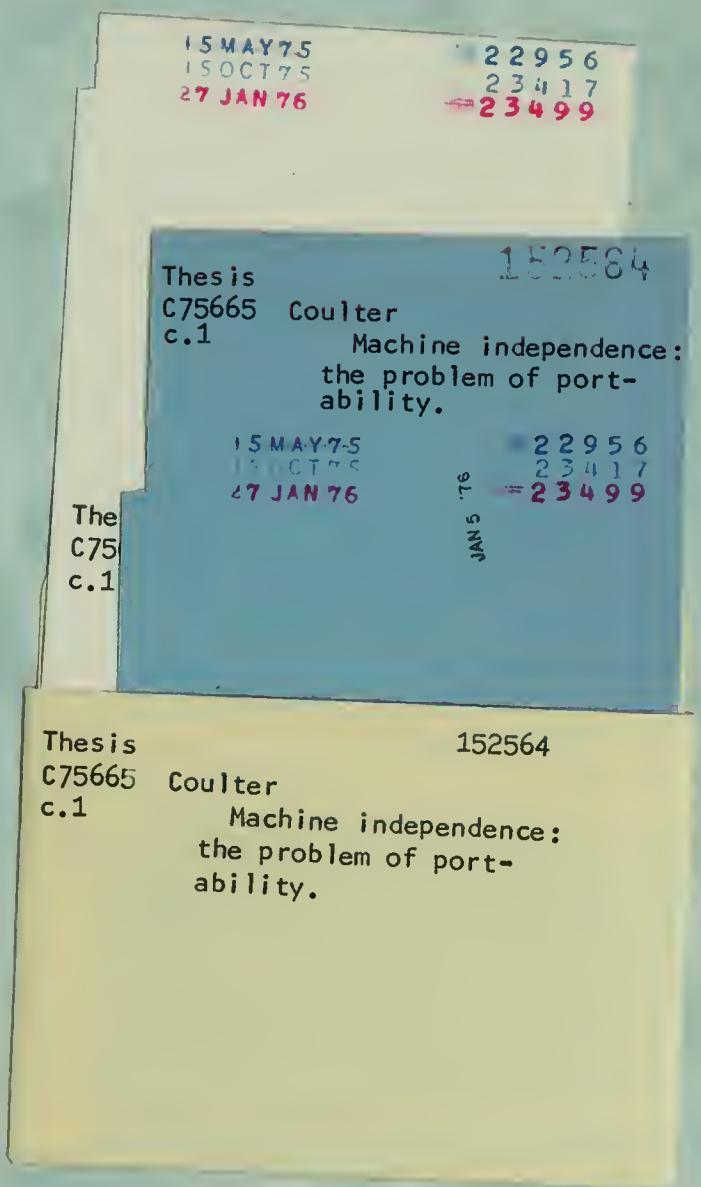
5. Parnas, D. L., "A Technique for Software Module Specification with Examples," Communication of the ACM, Volume 15, Number 5, p. 330-336, May 1972.

6. Sibley, R. A., "The SLANG System," Communications of the ACM, Volume 4, Number 1, p. 75-84, January 1961.

7. Steel, T. B., "UNCOL," Datamation, Number 1, p. 18-20, January/February, 1960.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chairman, Code 72 Computer Science Group Naval Postgraduate School Monterey, California 93940	1
4.	LT Edward C. Coulter, USN 841 E Bantam Road Tucson, Arizona	1
5.	LT Daniel J. Parker, USN 5722 Central Avenue Bonita, California	1
6.	Prof. Gary A. Kildall, Code 72Kd Computer Science Group Naval Postgraduate School Monterey, California 93940	1



15 MAY 75 22956
15 OCT 75 23417
27 JAN 76 23499

Thesis 152564
C75665 Coulter
c.1 Machine independence: the problem of portability.

15 MAY 75 22956
15 OCT 75 23417
27 JAN 76 23499
JAN 5 '76

The
C75
c.1

Thesis 152564
C75665 Coulter
c.1 Machine independence: the problem of portability.